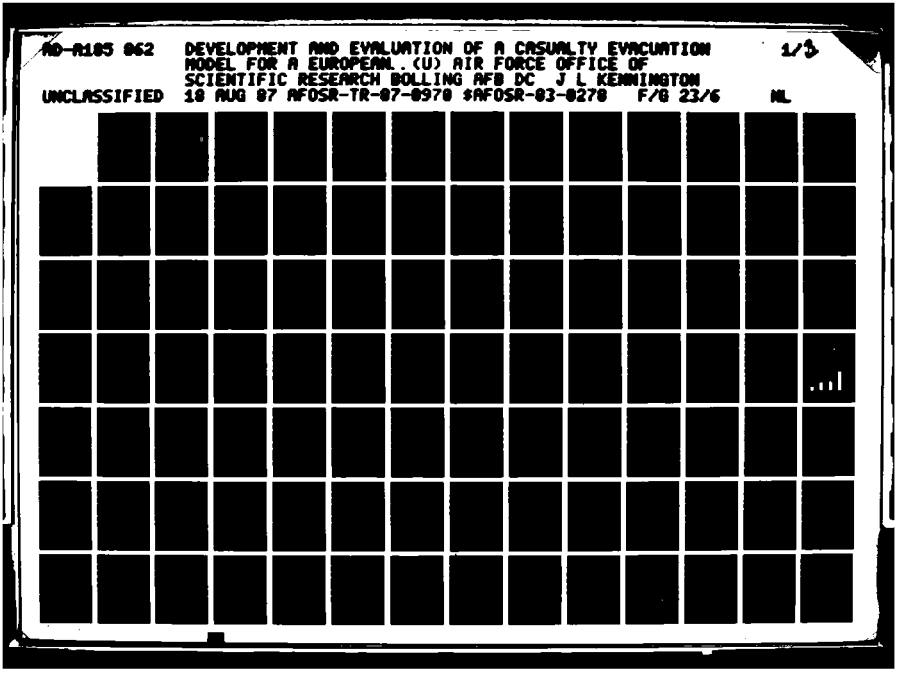
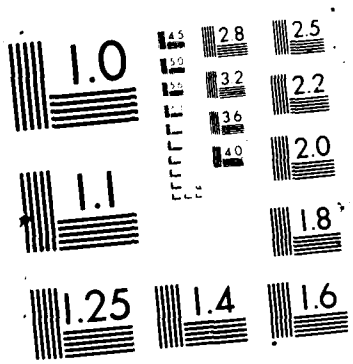

AD-A185 062 DEVELOPMENT AND EVALUATION OF A CASUALTY EVACUATION MODEL FOR A EUROPEAN.. (U) AIR FORCE OFFICE OF SCIENTIFIC RESEARCH BOLLING AFB DC J L KENNINGTON
1/3
UNCLASSIFIED 10 AUG 87 AFOSR-TR-87-0970 $AFOSR-83-0270 F/G 23/6 NL


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

AFOSR-TR- 87-0970

DEVELOPMENT AND EVALUATION OF A CASUALTY EVACUATION MODEL FOR A EUROPEAN CONFLICT

by

Jeffery L. Kennington

for

Air Force Office of Scientific Research
Bolling Air Force Base, DC 20332-6448

and

Headquarters Military Airlift Command
Scott Air Force Base, Illinois 62225

Final Technical Report

August 18, 1987

Contract No. AFOSR-83-0278

unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-0970
6a. NAME OF PERFORMING ORGANIZATION: SMRL (6b. OFFICE SYMBOL (If applicable))	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NM
6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code): AFOSR/NM, Bldg. 410, Bolling, AFB 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR (8b. OFFICE SYMBOL (If applicable))	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-83-0210
8c. ADDRESS (City, State, and ZIP Code): AFOSR, Bldg. 410, Bolling AFB, DC 20332-6448	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO A1; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Development and Evaluation of a Casualty Evacuation Model for a European Conflict (unclassified)

12. PERSONAL AUTHOR(S)
Dr. Jeffery L. Kennington

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 10/1/83 TO 12/31/86	87/8/17	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This project assisted the Operations Research Division, DCS/Plans, Headquarters MAC in the development of and solution algorithm for the Casualty Evacuation Model. In addition, we continued our investigation into new and improved algorithms for optimization on both serial and parallel computers.

DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Samual Rankin	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL: NM

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE: unclassified

EXECUTIVE SUMMARY

The research conducted under contract AFOSR 83-0278 is reported in seven technical reports corresponding to Chapters 1 through 7 in this report. A brief description of each study follows:

CHAPTER 1 USING TWO SEQUENCES OF PURE NETWORK PROBLEMS TO SOLVE THE MULTICOMMODITY NETWORK FLOW PROBLEM

Summary: This paper presents a new algorithm for solving large multicommodity network flow problems. The work was motivated by the Casualty Evacuation Model originally developed by Lt. Col. Dennis McLain. Captain Robert Chmielewski continued this activity and eventually a modification of this model was solved by the P.I. and Captain Chmielewski on a CDC Cyber 205. All of this activity was directed by Mr. Thomas Kowalsky of DSC/Plans of MAC Headquarters.

Publication Status: This work has not been submitted for publication.

Background: This was the dissertation research of Dr. Ellen Allen.

CHAPTER 2 NETWORKS WITH SIDE CONSTRAINTS: AN LU FACTORIZATION UPDATE

Summary: An important class of mathematical programming models which are frequently used in logistics studies is the model of a network problem having additional linear constraints. A specialization of the primal simplex algorithm which exploits the network structure can be applied to this problem class. This specialization maintains the basis as a rooted spanning tree and a general matrix called the working basis. This paper presents the algorithms which may be used to maintain the inverse of this working basis as an LU factorization, which is the industry standard for general linear programming software. Our specialized code exploits not only the network structure but also the sparsity characteristics of the working basis. Computational experimentation indicates that our LU implementation results in a 50 percent savings in the non-zero elements in the eta file, and our computer codes are approximately twice as fast as MINOS and XMP on a set of randomly generated multicommodity network flow problems.

Publication Status: Published in The Annals of the Society of Logistics Engineers, 1, 1, (1986), 66-85.

Background: This work is a summary of the dissertation research of Dr. Keyvan Farhangian.

CHAPTER 3 THE FREQUENCY ASSIGNMENT PROBLEM: A SOLUTION VIA NONLINEAR PROGRAMMING

Summary: This paper gives a mathematical programming model for the problem of assigning frequencies to nodes in a communications network. The objective is to select a frequency assignment which minimizes both cochannel and adjacent-channel interference. In addition, a design engineer has the option to designate key links in which the avoidance of jamming due to self interference is given a higher priority. The model has a nonconvex quadratic objective function, generalized upper-bounding constraints, and binary decision variables. We developed a special heuristic algorithm and software for this model and tested it on five test problems which were modifications of a real-world problem. Even though most of the test problems had over 600 binary variables, we were able to obtain a near optimum in less than 12 seconds of CPU time on a CDC Cyber-875.

Publication Status: Published in Naval Research Logistics, 34, (1987), 133-139.

Background: This was our first application in the communications area.

CHAPTER 4 A GENERALIZATION OF POLYAK'S CONVERGENCE RESULT FOR SUBGRADIENT OPTIMIZATION

Summary: This paper generalizes a practical convergence result first presented by Polyak. This new result presents a theoretical justification for the step size which has been successfully used in several specialized algorithms which incorporate the subgradient optimization approach.

Publication Status: Published in Mathematical Programming, 37, 3, (1987) 309-318.

Background: The convergence theory presented in this paper was motivated by the good computational results achieved by Drs. Ellen Allen and Bala Shetty in their dissertations.

CHAPTER 5 THE EQUAL FLOW PROBLEM

Summary: This paper presents a new algorithm for the solution of a network problem with equal flow side constraints. The solution technique is motivated by the desire to exploit the special structure of the side constraints and to maintain as much of the characteristics of pure network problems as possible. The proposed algorithm makes use of Lagrangean relaxation to obtain a lower bound and decomposition by right-hand-side allocation to obtain upper bounds. The Lagrangean dual serves not only to provide a lower bound used to assist in termination criteria for the upper bound, but also allows an initial allocation of equal flows for the upper bound. The algorithm has been tested on problems with up to 1500 nodes and 6000 arcs. Computational experience indicates that solutions whose objective function value is well within 1% of the optimum can be obtained in 1%-65% of the MPSX time depending on the amount of imbalance inherent

in the problem. Incumbent integer solutions which are within 99.99% feasible and well within 1% of the proven lower bound are obtained in a straightforward manner requiring, on the average, 30% of the MPSX time required to obtain a linear optimum.

Publication Status: This paper has been accepted for publication in the European Journal of Operations Research.

Background: This work is a summary of the dissertation research of Dr. Bala Shetty.

CHAPTER 6 A PARALLELIZATION OF THE SIMPLEX ALGORITHM

Summary: This paper presents a parallelization of the simplex method for linear programming. Current implementations of the simplex method on sequential computers are based on a triangular factorization of the inverse of the current basis. An alternative decomposition designed for parallel computation, called the quadrant interlocking factorization, has previously been proposed for solving linear systems of equations. This research presents the theoretical justification and algorithms required to implement this new factorization in a simplex-based linear programming system.

Publication Status: This paper has been submitted for publication and is currently under review.

Background: This paper is a summary of the dissertation research of Dr. Hossam Zaki.

CHAPTER 7 MINIMAL SPANNING TREES: A COMPUTATIONAL INVESTIGATION OF PARALLEL ALGORITHMS.

Summary: The objective of this investigation is to computationally test parallel algorithms for finding minimal spanning trees. Computational tests were run on a single processor using Prim's, Kruskal's and Boruvka's algorithms. Our implementation of Prim's algorithm is superior for high density graphs, while our implementation of Boruvka's algorithm is best for sparse graphs. Implementations of parallel versions of both Prim's and Boruvka's algorithms were tested on a twenty-cpu Balance 21000. For the environment in which a minimum spanning tree problem is a subproblem within another algorithm, the parallel implementation of both Boruvka's and Prim's algorithms produced speedups of three and five on five and ten processors, respectively. The one-time overhead for process creation negates most, if not all of the benefits for solving a single minimum spanning tree subproblem.

Publication Status: This paper has been submitted for publication and is currently under review.

Background: This is our first computational investigation which has been completed since the parallel computer arrived at Southern Methodist University.

CHAPTER 1

USING TWO SEQUENCES OF PURE NETWORK PROBLEMS TO SOLVE THE MULTICOMMODITY NETWORK FLOW PROBLEM

A Dissertation Presented to the Graduate

Faculty of the School of Engineering

and Applied Science

of

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

with major in

Operations Research

by

Ellen Parker Allen

B.A.S., Southern Methodist University, 1975

M.S.O.R., Southern Methodist University, 1981

May 1985

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

INTRODUCTION

This dissertation presents a new technique for solving very large multicommodity network flow problems. The specific application which motivated this work originated with the United States Air Force and was first presented to us by Lt. Col. Dennis McLain, the Assistant Director of Operations Research for the Military Airlift Command at Scott Air Force Base. The problem is an extremely large casualty evacuation model to be used by the Air Force in forming a plan for the evacuation of wartime casualties. This plan would be implemented in case of a European military conflict involving United States troops. Lt. Col. McLain was the first to model this problem as a multicommodity network flow problem where the commodities correspond to the various types of wounds. The nodes represent such entities as European bases and United States medical facilities, and the arcs represent specific aircraft flights. (A complete description of this problem is given in Section 1.3.) This problem is far too large to be solved by any known existing computer codes. In addition, since many of the data are only rough estimates (the number of casualties of various types expected at given locations), an exact technique is not called for. Instead a technique is needed to discover a guaranteed ε-optimum for any given $\varepsilon > 0$.

This is precisely what our technique accomplishes. It generates successively better upper and lower bounds on the optimum, stopping when the two bounds are within a prescribed tolerance. We exploit the multicommodity network structure in both the lower and upper bound routines so that only a single commodity minimum cost network flow optimizer is needed. EVAC, the computer code which implements our technique, has been used to solve a series of test problems in less time and requiring less memory than MCNF, a specialized multi-commodity network flow problem solver. In addition EVAC is capable of solving very large problems which MCNF is unable to solve.

1.1 Notation and Conventions

The notational conventions employed throughout this work are described in this section. Matrices are denoted by upper case Latin letters. The element of a matrix, A, which appears in the i^{th} row and j^{th} column is indicated by A_{ij}. The symbol I is used to denote an identity matrix with dimension appropriate to the context. Lower case Latin and Greek letters are used to denote vectors. The symbol 0 is used to represent a vector of zeroes with dimension appropriate to the context. The unit vector, whose only non-zero component is a one in the jth position, is denoted e_j. Subscripts are used to indicate individual components of a vector, or as an index to indicate which of

a sequence of related vectors is meant. Superscripts on vectors correspond to individual commodities. Note that vectors are considered to be row vectors or column vectors as appropriate to the context; that is, no special notation is used to indicate the transpose of a vector. The inner product of two vectors, x and y, is denoted simply by xy. The notation $||x||$ is used to express the Euclidian norm, $(xx)^{1/2}$. Scalars are written as lower case Greek or Latin letters.

Euclidean n-dimensional space is denoted R^n. Functions are written as lower case Latin letters, and functional values have their arguments in parentheses. For example g(y) is used to denote the function g evaluated at the point y. The one exception to this convention is the projection operation described in Chapter III. In this case P[x] is used to express the projection of x onto the specified region.

Upper case Greek letters denote sets, with the exception that $\partial g(y)$ is used to denote the set of subgradients of a function g at a point y in the domain of g. The symbol ε is used as the set inclusion symbol and as a termination tolerance.

We use MAX{S} to denote the largest element of a set S; similarly MIN{S} indicates the smallest element of S. The symbol ∞ is used for infinity, and ■ denotes the end of a proof. All other notation is standard.

1.2 Problem Definition

A network is composed of two entities: nodes and arcs. The arcs may be viewed as undirectional means of commodity transport, and the nodes may be thought of as locations or terminals connected by the arcs and served by whatever physical means of transport are associated with the arcs. We limit our consideration to networks with finite numbers of nodes and arcs. For a given network we denote the number of nodes by m and the number of arcs by n. We impose an ordering on the nodes and arcs so as to put them in a one-to-one correspondence with the integers {1,...,m} and {1,...,n}, respectively. The structure of a given network may be described, then, by an m x n matrix called a node-arc incidence matrix. Such a matrix, A, is defined in this way:

$$A_{ij} = \begin{cases} +1, & \text{if arc } j \text{ is directed away from node } i \\ -1, & \text{if arc } j \text{ is directed toward node } i \\ 0, & \text{otherwise.} \end{cases}$$

Additionally, for a multicommodity network, we are concerned with more than one type of item (commodity) flowing through the arcs. We order these commodities to correspond to the integers {1,...,K}.

We define the following quantities to be used in the formulation of the multicommodity network flow problem:

-- A is the m x n node-arc incidence matrix corresponding to the underlying network.

-- x^k is an n vector of decision variables for k = 1,...K. Note that x^k_j represents the amount of flow of commodity k on arc j.

-- c^k is an n vector of unit costs for k = 1,...,K. So c_j^k denotes the cost for one unit of flow of commodity k on arc j.

-- r^k is an m vector of requirements for k = 1,...,K, so that r_i^k denotes the required number of units of commodity k at node i. If $r_i^k < 0$ then node i is said to be a demand node for commodity k with demand = $|r_i^k|$. If $r_i^k > 0$ then node i is said to be a supply node for commodity k with supply = r_i^k. And if $r_i^k = 0$ then node i is said to be a transshipment node for commodity k.

-- u is an n vector of mutual arc capacities. That is, the total flow of all the commodities combined for arc j cannot exceed u_j.

-- v^k is a n vector of arc capacities for commodity k (k = 1,...,K). v_j^k, then, represents an upper bound on the flow of commodity k on arc j.

We sometimes refer to the entire vector of decision variables $(x^1,...,x^K)$ as simply x. Similarly we use c, r, and v to denote the entire vector of costs, requirements and upper bounds, respectively.

Using these ideas, we may formulate the multicommodity network flow problem for a given network with m nodes, n arcs, and K commodities as follows:

$$\text{Minimize} \quad \sum_k c^k x^k$$

$$\text{Subject to } Ax^k = r^k, \quad k = 1,...,K \qquad \text{(MP)}$$

$$\sum_k x^k \leq u$$

$$0 \leq x^k \leq v^k, \quad k = 1,...,K.$$

1.3 The Casualty Evacuation Model

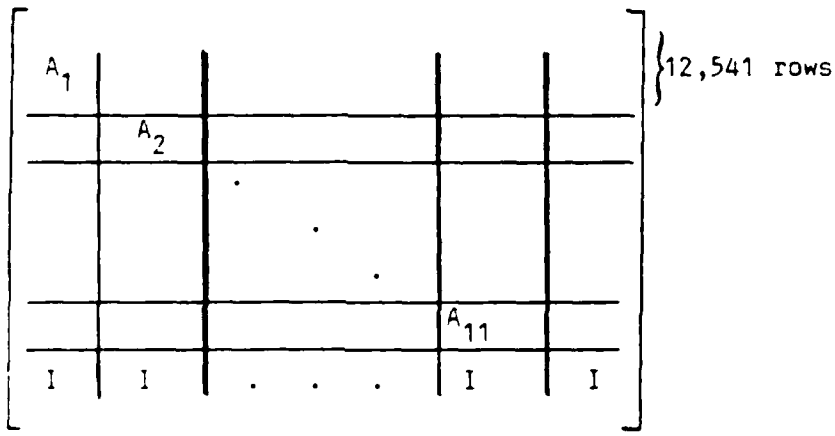
A large European military conflict involving U.S. Armed Forces could result in more casualties than could be effectively handled in European medical facilities. To alleviate this overcrowding, the Department of Defense plans to implement the following evacuation policy:

> "During the first 30 days of a conflict, if a wounded soldier cannot be returned to duty within 15 days, then he will be evacuated to a medical facility in the United States. In the next 30 days the limit on treatment time is increased to 30 days."

Given a scenario concerning such a conflict (i.e. the number and locations of wounded and the types of wounds), this evacuation problem may be modelled as a multicommodity network flow problem. Lt. Col. Dennis McLain was the first to model the problem in this way. In Lt. Col. McLain's model the nodes correspond to 9 European recovery bases and 95 United States locations. The arcs correspond to aircraft flights connecting European and U.S. facilities. The commodities are 11 different patient types.

In order to enforce a capacity on a given facility, it is necessary to duplicate the corresponding node using the capacity as an upper bound on the arc between the duplicate nodes. For example, if node A represents a hospital with 300 beds, then we substitute two

nodes, A1 and A2, along with an arc whose capacity is 300. Further, it is necessary to include 60 copies of the entire network, one for each of the 60 one-day time periods. Additional arcs are created to link each time period to the next. The model includes a dummy "sink" node for each time period and one "super sink" node, along with capacitated arcs to allow patients who have recovered to exit from the system. These considerations produce a very large model. The dimensions of the constraint matrix are shown below:



where $A_1 = \ldots = A_{11}$. The row dimension of this model is over 137,000, which is far beyond the scope of any known existing computer code. To put these figures in perspective, we note that Kennington reports that the largest models he has solved using his primal partitioning code, MCNF, have been on the order of 3000 rows [2].

Our plan has been to develop a specialized solution procedure which would solve a scaled-down version of Lt. Col. McLain's model. We anticipate aggregation of the data, possibly using some of the following ideas:

(1) Aggregation of the time periods. Note that simply using 3-day time periods instead of 1-day time periods reduces the problem size to around 46,000 rows.

(2) Aggregation of similar patient types.

(3) Aggregation of U.S. medical facilities so that facilities which are located within a given number of miles of one another are treated as one node.

At the writing of this dissertation we have not yet received any large test problems from the Air Force. As a result, we are unable to report on the problem size limitations of our technique. However, in an attempt to test our software on a relatively large problem, we solved a randomly generated test problem with around 9,000 rows. (See Chapter 4 for the details of this problem.) This is the largest problem we have attempted so far.

1.4 Accomplishments of This Investigation

This dissertation proposes a new technique for solving extremely large multicommodity network flow problems. Our method involves generating upper bounds on the optimal objective value by partially solving the problem using a resource-directive decomposition technique, and generating lower bounds on the optimal objective value by partially solving a Lagrangian dual of the problem. Both the upper and lower bound routines exploit the network structure of the problem, decomposing it by commodities and solving the resulting pure network problems. In the limit both bounds must converge to the optimal objective function value; in practice we stop when the difference between the two bounds is within some termination tolerance.

Whether solving for lower bounds or for upper bounds a subgradient optimization technique is used. At each iteration this procedure requires the computation of a subgradient, the selection of a step size, and a projection operation. In Section 3.1, we obtain a new convergence result for a particular class of subgradient procedures. Then, in Section 3.2, we introduce a new heuristic, closely related to the subgradient optimization procedure, which has worked well for our test problems.

Our technique has been tested on randomly generated test problems and on one problem which was formulated specifically to represent the class of evacuation planning problems for which the code was developed. In addition, the same set of test problems was solved by MCNF [51], a general purpose multicommodity network flow problem solver which uses a primal partitioning scheme. Computation times for both codes are presented. Our code used an average of 68% of the time needed by MCNF, performing significantly better on the problems with fewer commodities. In addition our code required on the order of 1/K the amount of main memory for a K-commodity problem, so it can solve significantly larger problems than MCNF.

CHAPTER II

A SURVEY OF RELATED LITERATURE

In this chapter we present an overview of the existing work on which this dissertation is based. Section 2.1 deals with the work that has been done in the area of pure network models. Then in Section 2.2 we address the broader area of multicommodity network methods. Since our algorithm involves a subgradient optimization technique, both in the Lagrangian dual portion and in the resource-directive decomposition routine, we provide some references involving subgradient optimization in Section 2.3

2.1 Pure Networks

Network problems are linear programming problems with node-arc incidence matrices as their constraint matrices. Within this class, known formally as minimal cost network flow problems, there are several variations including transportation problems, transshipment problems, assignment problems, maximal flow problems, and shortest path problems.

Ideas for solution of network problems can be traced at least as far back as 1939, to the work of Professor Leonid Kantorovich [41]. Kantorovich, along with Professor Tjalling C. Koopmans received the Nobel Prize in Economic Science in 1975, for contributions to the theory of optimum allocation of scarce resources. Koopmans and Reiter

[54] and Frank L. Hitchcock [42], working independently, were the first to formulate the transportation problem. The mid-fifties saw a surge of interest and work in the areas of network algorithms. It was around this time at Alex Orden [59] generalized the transportation model to allow transshipment points. Lester Ford and Delbert Fulkerson [22] [20] formulated and investigated solution techniques for the maximal flow problem and the minimal cost network flow problem. The specialized algorithms that have been developed for solving network problems may be classified into two groups: primal-dual techniques, and specializations of the primal simplex algorithm. Primal-dual methods for solving networks began with Harold Kuhn's Hungarian Algorithm for the assignment problem [55] and culminated in Fulkerson's Out-of-Kilter Algorithm [23]. Primal simplex based techniques originated with the work of Professor George Dantzig [17] and continued through Ellis Johnson's 1965 paper [47]. The basis for Johnson's work can be traced to the work of Dantzig [18] and Charnes and Cooper [14].

Since that time much work has been done in the area of solution techniques, and computational advances have been made by the development of more efficient data structures. The credit for much of this work goes to Professors Fred Glover and Darwin Klingman and their colleagues at the University of Texas. This is evidenced by such papers as Barr, Glover and Klingman [9] [10], Glover, Hultz and Klingman [26] [25], Glover, Karney and Klingman [27], Glover, Karney, Klingman and Napier [28], Glover and Klingman [29] [31] [30], Glover, Klingman and Stutz [32], and Karney and Klingman [49]. Others who have contributed to the research include Srinivasan and Thompson [63] [64],

Bradley, Brown, and Graves [13], and Mulvey [57] [58]. In addition significant work has been performed by Professors Jeff Kennington, Richard Barr, and Richard Helgason of Southern Methodist University as seen in such works as [3], [41], and [52].

Today network algorithms have been demonstrated to solve linear network problems 50 times faster than general linear programming algorithms [6]. Additionally a computer implementation of such a technique may require only half the memory of the general L.P. package [6]. These advances are due to the efficient data structures which have been developed to allow a basis for a network problem to be stored as a rooted spanning tree on the nodes in the network. Using this idea all the simplex computations such as pricing, ratio test, and updates can be performed via labelling algorithms on the basis tree. This eliminates the need to store the basis inverse in factored form.

2.2 Multicommodity Networks

Multicommodity network flow problems are problems in which several different types of items (commodities) must share arcs in a capacitated network. Each solution technique for multicommodity network models can be classified as one of two main types of algorithms: partitioning algorithms and decomposition algorithms.

2.2.1 Partitioning Algorithms

Partitioning algorithms are specializations of the simplex method which exploit the multicommodity network structure by partitioning the basis into more than one part. In one part advantage is taken of the

special network structure. Those who have studied primal partitioning algorithms include Kennington [50], Helgason and Kennington [40], Ali, Helgason, Kennington, and Lall [4], Hartman and Lasdon [36] [35], Maier [56], and Saigal [61]. Ali and Kennington [6], in their computational research, reported solution times averaging 5 times faster than general linear programming codes. A dual partitioning method was proposed by Grigoriadis and White [34]. A primal-dual partitioning scheme was developed by Jewell [46]. In addition a factorization technique was proposed by Graves and McBride [33]. MCNF, the multicommodity network code with which we compared our solution times, is a primal partitioning program.

2.2.2 Decomposition Algorithms

Decomposition schemes seek to solve the problem by decomposing it into several smaller subproblems, each of which takes the form of a pure minimum cost network flow problem. A master program coordinates the solution process. Decomposition procedures for the multicommodity network flow problem fall into two categories: price-directive schemes and resource-directive schemes.

Price-directive decomposition is based on the well-known research of Dantzig and Wolfe [19]. In a price-directive approach, the K-commodity problem is decomposed into K single commodity problems. The master program then uses the simplex method while the subproblems test for optimality and select candidates to enter the basis of the master problem. Ford and Fulkerson [21] were the first to develop this

approach for solving multicommodity network flow problems. Tomlin [67] was the first to develop a computer code implementing this technique. Others who have studied price-directive decomposition schemes are Jarvis [43], Jarvis and Keith [44], Chen and DeWald [15], and Jarvis and Martinez [45]. Price-directive approaches for generalizations of this problem have been proposed by Cremeans, Smith, and Tyndall [16], Swoveland [65] [66], Weigel and Cremeans [68], and Wollmer [69].

Resource-directive decomposition schemes decompose the problem by commodities, and the master problem systematically distributes the mutual arc capacities among the commodities. At each iteration the optimal solutions to the single commodity subproblems are used to compute a new set of allocations. Robacker [60] was the first to suggest this approach for multicommodity network problems. Research on this technique has been presented by Swoveland [65], Assad [8], Ali, Helgason, Kennington and Lall [3], and Kennington and Shalaby [53].

2.3 Subgradient Optimization

The subgradient optimization technique was first proposed by Shor [62] in 1964. Since that time subgradient algorithms have been applied to many different optimization problems. Held and Karp [37] and Held, Wolfe and Crowder [38] made use of the approach in solving the symmetric travelling salesman problem. Bazaraa and Goode [11] applied the algorithm to the asymmetric travelling salesman problem. Subgradient methods have been used to solve the assignment problem [38]. Glover, Glover and Martinson [24] applied a subgradient technique to

solve a special network with side constraints, and Ali and Kennington [7] made use of it in research involving the m-travelling salesman problem.

CHAPTER III

THE ALGORITHM

Here we present a new solution technique for the multicommodity network flow problem. This technique involves finding successively better upper and lower bounds on the optimal objective function value. The algorithm terminates whenever the two bounds are within a prescribed tolerance or when it can be shown that the current solution is an exact optimum.

Lower bounds are generated by partially solving a Lagrangian dual. At each iteration a Lagrangian relaxation of the original problem is solved; since these relaxations decompose on commodities, only a (single-commodity) minimum cost network flow optimizer is needed. A subgradient direction is used to adjust the Lagrange multipliers for the next iteration.

Upper bounds are generated using a modification of the resource-directive decomposition technique first suggested by Robacker [60]. We introduce a specialization of the subgradient direction approach which was first applied to this class of problems by Held, Wolfe, and Crowder [38].

With minor restrictions on the step sizes we show that both the upper and lower bounds converge to the optimal objective value of the original multicommodity network flow problem. Hence in the limit the algorithm will converge to an exact optimum. In practice we seek a near-optimum.

3.1 Subgradient Optimization

Let us first consider the general subgradient algorithm for optimization of convex functions; later we will present specializations of the technique for the upper and lower bound problems. Consider the nonlinear programming problem

Minimize $g(y)$

Subject to $y \in \Gamma$

where g is a real valued function that is convex over the compact, convex, nonempty set Γ. A vector π is called a subgradient of g at a point x if

$$g(y) - g(x) \geq \pi(y - x) \text{ for all } y \in \Gamma.$$

Note that if g is differentiable at x, the only subgradient at x is the gradient. We denote the set of all subgradients of g at x by $\partial g(x)$.

The subgradient algorithm proceeds in this manner: Given a point x in Γ, find a subgradient of g at x, obtain a new point by moving a given step size in the negative subgradient direction, and finally project this new point back onto Γ. This projection operation takes a point x and finds the point in Γ that is "closest" to x with respect to the Euclidean norm. We denote the projection of x onto Γ by $P[x]$. Using this notation we present the general subgradient optimization algorithm for minimizing a convex function g [52].

ALGORITHM 3.1 SUBGRADIENT OPTIMIZATION ALGORITHM

Step 0 (Initialization)

Let y_0 be any element of Y. Select a set of step sizes, $s_1, s_2, s_3, \ldots$, and set $i \leftarrow 0$.

Step 1 (Find Subgradient)

Let $\pi_i \in \partial g(y_i)$. If $\pi_i = 0$ terminate with y_i optimal.

Step 2 (Move to New Point)

Set $y_{i+1} \leftarrow P[y_i - s_i \pi_i]$. Set $i \leftarrow i + 1$. Return to step 1.

Let us now turn our attention to the selection of step sizes. Several ideas for choosing step sizes have been proposed. These typically involve a sequence of constants, $\{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ which satisfy the following conditions:

$$\lambda_i > 0, \quad \text{for all } i,$$

$$\lim_{i \to \infty} \lambda_i = 0, \quad \text{and} \tag{3.1}$$

$$\sum_i \lambda_i = \infty .$$

The subgradient algorithm can be shown to converge when any of the following three formulae are used for determining step sizes [52]:

$$\text{(i)} \quad s_i = \lambda_i ,$$

$$\text{(ii)} \quad s_i = \lambda_i / ||\pi_i||^2 , \tag{3.2}$$

$$\text{(iii)} \quad s_i = \lambda_i [g(y_i) - g^*] / ||\pi_i||^2$$

where g^* denotes the optimal objective value.

Propositions 3.1, 3.2, and 3.3 may be found in Kennington and Helgason [52], and are given here as necessary preliminary results.

Proposition 3.1 [52]

Let $y \in \Gamma$, and let $x \in R^n$. Then $(x-P[x])(y-P[x]) \leq 0$.

Proof

Choose α so that $0<\alpha<1$. Since Γ is convex, $\alpha y+(1-\alpha)P[x] \in \Gamma$. By the definition of $P[x]$, $||x-P[x]|| \leq ||x-(\alpha y+(1-\alpha)P[x])||$. Thus

$$\begin{aligned} ||x-P[x]||^2 &\leq ||x-(\alpha y+(1-\alpha)P[x])||^2 \\ &= ||x-P[x]-\alpha(y-P[x])||^2 \\ &= ||x-P[x]||^2+\alpha^2||y-P[x]||^2 -2\alpha(x-P[x])(y-P[x]). \end{aligned}$$

Then $(x-P[x])(y-P[x]) \leq ||y-P[x]||\alpha/2$. And, since α can be taken arbitrarily close to 0,

$$(x-P[x])(y-P[x]) \leq 0 . \quad \blacksquare$$

Proposition 3.2 [52]

Let $x, y \in R^n$. Then $||P[x]-P[y]|| \leq ||x-y||$.

Proof

Case 1: Supose $P[x] = P[y]$. Then

$$||P[x]-P[y]|| = 0 \leq ||x-y||.$$

Case 2: Suppose $P[x] \neq P[y]$. Then since $P[x] \in \Gamma$, and $P[y] \in \Gamma$, from Proposition 3.1 we have that

$$(x-P[x])(P[y]-P[x]) \leq 0$$

and

$$(y-P[y])(P[x]-P[y]) \leq 0.$$

We may rewrite the above inequalities as

$$x(P[y]-P[x])-P[x]P[y]+||P[x]||^2 \leq 0$$

and

$$y(P[x]-P[y])-P[y]P[x]+||P[y]||^2 \leq 0.$$

Adding these inequalities, we obtain

$$(x-y)(P[y]-P[x])+||P[y]-P[x]||^2 \leq 0.$$

Then from the Cauchy-Schwartz inequality,

$$-(x-y)(P[y]-P[x]) \leq ||x-y||\ ||P[y]-P[x]||.$$

Thus

$$||P[y]-P[x]||^2 \leq ||x-y||\ ||P[y]-P[x]||.$$

And since $P[x] \neq P[y]$,

$$||P[x]-P[y]|| \leq ||x-y||. \quad \blacksquare$$

Proposition 3.3 [52]

If $\eta_i \neq 0$, then, for any $y \in T$,

$$||y_{i+1}-y||^2 \leq ||y_i-y||^2 + s_i^2||\eta_i||^2+2s_i\eta_i(y-y_i).$$

Proof

Let i be any iteration of the subgradient algorithm. Suppose $\eta_i \neq 0$. Let $y \in T$. Then, by Proposition 3.2,

$$||P[y_i-s_i\eta_i]-P[y]||^2 \leq ||y_i-s_i\eta_i-y||^2$$

$$= ||y_i-y||^2 + s_i^2||\eta_i||^2 + 2s_i\eta_i(y-y_i).$$

Since $P[y]=y$ and $P[y_i-s_i\eta_i] = y_{i+1}$, we have that

$$||y_{i+1}-y||^2 \leq ||y_i-y||^2 + s_i^2||\eta_i||^2 + 2s_i\eta_i(y-y_i) . \quad \blacksquare$$

Our main convergence result is for the particular step size scheme:

$$s_i = \lambda_i[g(y_i) - \bar{g}]/||\eta_i||^2$$

where $\bar{g}$ is a lower bound for the optimal objective and where we are at liberty to select bounds α and β for the $\{\lambda_i\}$ such that for each i, $0 < \alpha \leq \lambda_i \leq \beta < 2$.

Proposition 3.4

Let (i) $\bar{g}$ be a known lower bound for the optimal objective, g^*, with $g^* > \bar{g}$;

(ii) $\{\lambda_i\}$ be any infinite sequence such that for all i, $0 < \alpha \leq \lambda_i \leq \beta < 2$; and

(iii) $s_i = \lambda_i[g(y_i)-\bar{g}]/||\eta_i||^2$.

If there is a constant C such that for all i, $||\eta_i|| \leq C$, and if $\gamma > 0$ is given, then there is some n such that $g(y_n) \leq g^* + [\beta/(2-\beta)](g^*-\bar{g})+\gamma$.

Proof

Let $\gamma > 0$ be given. Let (i), (ii), and (iii) hold. Let y^* be an optimal point, and for all i, $||\eta_i|| \leq C$. Suppose, contrary to the desired result, that for all n, $g(y_n) > g^* + [\beta/(2-\beta)](g^*-\bar{g})+\gamma$. Then, by Proposition 3.3,

$$||y_{i+1}-y^*||^2 \leq ||y_i-y^*||^2+\lambda_i^2[g(y_i)-\bar{g}]^2/||\eta_i||^2$$

$$+ 2\lambda_i\{[g(y_i)-\bar{g}]/||\eta_i||^2\}\eta_i(y^*-y_i)$$

$$\leq ||y_i-y^*||^2+\lambda_i^2[g(y_i)-\bar{g}]^2/||\eta_i||^2$$

$$+ 2\lambda_i\{[g(y_i)-\bar{g}]/||\eta_i||^2\}[g^*-g(y_i)],$$

since $\eta_i \in \partial g(y_i)$.

Since $\beta \geq \lambda_i > 0$, then $\beta\lambda_i \geq \lambda_i^2$. So,

$$||y_{i+1}-y^*||^2 \leq ||y_i-y^*||^2+\beta\lambda_i[g(y_i)-\bar{g}]^2/||\eta_i||^2$$

$$+ 2\lambda_i\{[g(y_i)-\bar{g}]/||\eta_i||^2\}[g^*-g(y_i)]$$

$$= ||y_i-y^*||^2+(2-\beta)\lambda_i\{[g(y_i)-\bar{g}]/||\eta_i||^2\}$$

$$[(g^*-g(y_i))+(\beta/(2-\beta))(g^*-\bar{g})].$$

Since $g(y_i) > g^*+(\beta/(2-\beta))(g^*-\bar{g})+\gamma$, then $-\gamma > g^*-g(y_i) + (\beta/(2-\beta))(g^*-\bar{g})$. So,

$$||y_{i+1}-y^*||^2 < ||y_i-y^*||^2-(2-\beta)\lambda_i[g(y_i)-\bar{g}]\gamma/||\eta_i||^2.$$

Since $g^* \leq g(y_i)$, $\alpha \leq \lambda_i$, and $||\eta_i|| \leq C$, then

$$||y_{i+1}-y^*||^2 < ||y_i-y^*||^2-[(2-\beta)\alpha(g^*-\bar{g})\gamma]/C^2. \qquad (3.3)$$

We can choose an integer N so large that

$$C^2||y_1-y^*||^2/(2-\beta)\alpha(g^*-\bar{g})\gamma < N.$$

Thus, since $2-\beta>0$ and $g^*-\bar{g}>0$,

$$N(2-\beta)\alpha(g^*-\bar{g})\gamma/C^2 > ||y_1-y^*||^2.$$

Adding together the inequalities obtained from (3.3) by letting i take on all values from 1 to N, we obtain

$$||y_{N+1}-y^*||^2 < ||y_1-y^*||^2-N(2-\beta)\alpha(g^*-\bar{g})\gamma/C^2 < 0,$$

a contradiction. ■

It is shown in [39] that when Γ is compact, g is continuous on some open set containing Γ, and $\partial g(y) \neq \phi$ for all $y \in \Gamma$, there exists a constant C such that $||\eta|| \leq C$ for all $y \in \Gamma$, and $\eta \in \partial g(y)$, so that the boundedness condition on the subgradients in Proposition 3.4 is easily met.

3.2 Generating Lower Bounds

In this section we present a technique for generating lower bounds for the multicommodity network flow problem. This technique involves partially solving the Lagrangian dual problem using a subgradient technique to update the Lagrange multipliers at each iteration.

Recall that the multicommodity network flow problem, MP, may be stated as follows:

$$\text{Minimize} \sum_k c^k x^k$$

$$\text{Subject to } Ax^k = r^k, \quad k = 1,\ldots,K \qquad \text{(MP)}$$

$$\sum_k x^k \leq u$$

$$0 \leq x^k \leq v^k, \quad k = 1,\ldots,K$$

where

A is an m x n node-arc incidence matrix,

c^k is an n vector of unit costs for k = 1,...,K,

r^k is an m vector of node requirements for k = 1,...,K,

u is an n vector of mutual arc capacities,

v^k is an n vector of individual commodity bounds for k=1,...,K,

x^k is an n vector of decision variables for k = 1,...,K,

and K is the number of commodities.

Consider a Lagrangian dual problem for MP, denoted by DP:

$$\text{MAX} \quad h(\lambda)$$
$$\lambda \geq 0$$

$$h(\lambda) = \text{MIN}[\ \sum_k c^k x^k + \lambda(\sum_k x^k - u): \qquad \text{(DP)}$$

$$Ax^k = r^k \ (k = 1,\ldots,K);\ 0 \leq x^k \leq v^k \ (k = 1,\ldots,K)]$$

where λ is an n vector of Lagrange multipliers.

First we show that any feasible solution for DP is a lower bound for MP.

Proposition 3.5 [12]

Let $\bar{x} = (\bar{x}^1, \bar{x}^2, \ldots, \bar{x}^K)$ be a feasible solution for MP. Let $\bar{\lambda}$ be a feasible solution for DP. Then $h(\bar{\lambda}) \leq c\bar{x}$.

Proof

Since $h(\bar{\lambda})$ is a minimum, and since $\bar{x}$ is feasible for MP, $h(\bar{\lambda}) \leq \sum_k c^k \bar{x}^k + \bar{\lambda}(\sum_k \bar{x}^k - u)$. Further since $\bar{\lambda}$ is feasible for DP and $\bar{x}$ is feasible for MP, then $\bar{\lambda}(\sum_k \bar{x}^k - u) \leq 0$.

Hence $h(\bar{\lambda}) \leq c\bar{x}$. ■

In addition to this result, Bazaraa and Shetty [12] have proved that if MP has an optimal solution, then DP has an optimal solution, and that their optimal objective function values are equal. As a result, we see that we may indeed solve (or partially solve) DP in order to obtain a lower bound for MP.

In order to justify using a subgradient optimization technique for solving DP, we must show that the objective function is concave and develop an expression for a subgradient.

Proposition 3.6

The real valued function h is concave over $\Lambda = \{\lambda : \lambda \in R^n;\ \lambda \geq 0\}$.

Proof

Let $\lambda^1 \geq 0$. Let $\lambda^2 \geq 0$. Let $0 < \alpha < 1$. Then

$$h(\alpha\lambda^1 + (1-\alpha)\lambda^2) = \text{MIN}[\sum_k c^k x^k + (\alpha\lambda^1 + (1-\alpha)\lambda^2)(\sum_k x^k - u):$$

$$Ax^k = r^k (k=1,\ldots,K); 0 \leq x^k \leq v^k\ (k=1,\ldots,K)]$$

$$= \text{MIN}[\alpha\sum_k c^k x^k + \alpha\lambda^1(\sum_k x^k - u) + (1-\alpha)\sum_k c^k x^k + (1-\alpha)\lambda^2(\sum_k x^k - u):$$

$$Ax^k = r^k (k=1,\ldots,K);\ 0 \leq x^k \leq v^k (k=1,\ldots,K)]$$

$$\geq \alpha\,\text{MIN}[\sum_k c^k x^k + \lambda^1(\sum_k x^k - u):$$

$$Ax^k = r^k (k=1,\ldots,K); 0 \leq x^k \leq v^k (k=1,\ldots,K)]$$

$$+ (1-\alpha)\ \text{MIN}\ [\sum_k c^k x^k + \lambda^2(\sum_k x^k - u)\ :$$

$$Ax^k = r^k (k=1,\ldots,K); 0 \leq x^k \leq v^k\ (k=1,\ldots,K)]$$

$= \alpha h(\lambda^1) + (1-\alpha)h(\lambda^2)$. Hence h is concave over Λ. ■

Proposition 3.7

Let $\bar{\lambda} \geq 0$. Let $\bar{x}$ represent an optimal value of x corresponding to $h(\bar{\lambda})$. Then $d = \sum_k \bar{x}^k - u$ is a subgradient of h at $\bar{\lambda}$.

Proof

Let $\hat{\lambda}$ be any other point in Λ with corresponding optimal decision variable values $\hat{x}$. Then

$$h(\hat{\lambda}) = \sum_k c^k \hat{x}^k + \hat{\lambda}(\sum_k \hat{x}^k - u)$$

$$\leq \sum_k c^{k-k}\hat{x} + \hat{\lambda}(\sum_k \bar{x}^k - u) \quad \text{(since } \hat{x} \text{ is optimal)}$$

$$= \sum_k c^{k-k}\hat{x} + \hat{\lambda}(\sum_k \bar{x}^k - u) + (\bar{\lambda}\sum_k \bar{x}^k - \bar{\lambda}\sum_k \bar{x}^k) + (\bar{\lambda}u - \bar{\lambda}u)$$

$$= \sum_k c^{k-k}\hat{x} + \bar{\lambda}(\sum_k \bar{x}^k - u) + (\sum_k \bar{x}^k - u)(\hat{\lambda} - \bar{\lambda})$$

$$= h(\bar{\lambda}) + d(\hat{\lambda} - \bar{\lambda}).$$

Therefore d is a subgradient of h at $\bar{\lambda}$. ■

We now present our algorithm for computing lower bounds for MP. Note that it is a specialization of the subgradient optimization algorithm for this problem, and its convergence follows as a maximization analog of Proposition 3.4.

ALGORITHM 3.2 LOWER BOUND ALGORITHM

Step 0 (Initialization)

Let UB be any upper bound on the solution to MP. Set $i \leftarrow 0$; $c \leftarrow 0$; $\alpha_0 \leftarrow 2$. Compute $y_0 \leftarrow h(\lambda_0)$ and let $x_0 = (x_0^1, \ldots, x_0^K)$ be the corresponding optimal values of the decision variables.

Step 1 (Find Subgradient)

Set $r_i \leftarrow \sum_k x_i^k - u$. If $r_i = 0$, stop with y_i optimal.

Step 2 (Move to New Point)

Set $s_i \leftarrow \alpha_i(UB - y_i)/||r_i||^2$. Compute the j^{th} component of λ_{i+1} as:

$$(\lambda_{i+1})_j \leftarrow \text{MAX}\{(\lambda_i + s_i r_i)_j, 0\}$$

Compute $y_{i+1} \leftarrow h(\lambda_{i+1})$ and let x_{i+1} be the corresponding optimal values of the decision variables. Set $\alpha_{i+1} \leftarrow \alpha_i/2$. Set $i \leftarrow i+1$. Return to step 1.

3.3 Generating Upper Bounds

Here we describe a procedure for generating upper bounds for the multicommodity network flow problem. This procedure is a specialization of the resource-directive decomposition (RDD) algorithm using a subgradient direction. First we describe the general RDD procedure; then we present our specialization.

The RDD technique produces a sequence of feasible solutions by distributing the mutual arc capacity among commodities in such a way that the solutions to the K individual subproblems provide a solution to the composite problem. At each iteration an allocation is made and the resulting K (single commodity) minimum cost network flow problems are solved. If the solution meets an optimality criterion then the procedure terminates; otherwise, a new allocation is made, and the process is repeated.

After introducing artificial variables, (a^k), MP becomes:

$$\text{Minimize } \sum_k c^k x^k + M \sum_k \underline{1} a^k$$

$$\text{Subject to } Ax^k + a^k = r^k \quad (k = 1,\ldots,K)$$

$$\sum_k x^k \leq u$$

$$0 \leq x^k \leq v^k \quad (k = 1,\ldots,K)$$

$$a^k \geq 0 \quad (k = 1,\ldots,K)$$

where M is a very large positive number and $\underline{1}$ is an m vector of all ones.

Let us restate the problem as:

$$\text{Minimize } z(y^1,\ldots,y^K)$$

$$\text{Subject to } z(y^1,\ldots,y^K) = \sum_k z^k(y^k) \qquad \text{(RP)}$$

$$\sum_k y^k = u$$

$$0 \leq y^k \leq v^k \quad (k = 1,\ldots,K)$$

where $z^k(y^k) = \text{MIN}\{c^k x^k + M\underline{1}a^k : Ax^k + a^k = r^k;\ 0 \leq x^k \leq y^k;\ a^k \geq 0\}$ for $k = 1,\ldots,K$. We shall refer to this formulation as RP. Note that $z^k(y^k) = \text{MAX}\{r^k \mu^k - y^k \nu^k : \mu^k A - \nu^k \leq c^k;\ \mu^k \leq M\underline{1};\ \nu^k \geq 0\}$, by duality theory.

In order to justify using a subgradient optimization technique we must show that $z(y^1,\ldots,y^K)$ is a convex function and develop an expression for a subgradient.

Proposition 3.8 [52]

The real valued function z is convex over $Y = \{y^1,\ldots,y^K : y^1 \geq 0;\ldots;y^K \geq 0\}$.

Proof

Let $(\bar{y}^1,\ldots,\bar{y}^K) \in Y$ and $(\hat{y}^1,\ldots,\hat{y}^K) \in Y$. Select α so that $0 \leq \alpha \leq 1$. Then

$$z[\alpha\bar{y}^1 + (1-\alpha)\hat{y}^1,\ldots,\alpha\bar{y}^K + (1-\alpha)\hat{y}^K]$$

$$= \sum_k z^k[\alpha\bar{y}^k + (1-\alpha)\hat{y}^k]$$

$$= \sum_k \text{MAX}\{r^k\mu^k - [\alpha\bar{y}^k + (1-\alpha)\hat{y}^k]\nu^k :$$

$$\mu^k A - \nu^k \leq c^k;\ \mu^k \leq M\underline{1};\ \nu^k \geq 0\}$$

$$= \sum_k \text{MAX}\{\alpha[r^k\mu^k - \bar{y}^k\nu^k] + (1-\alpha)[r^k\mu^k - \hat{y}^k\nu^k]:$$

$$\mu^k A - \nu^k \leq c^k;\ \mu^k \leq M1;\ \nu^k \geq 0\}$$

$$\leq \alpha\sum_k \text{MAX}\{r^k\mu^k - \bar{y}^k\nu^k:$$

$$\mu^k A - \nu^k \leq c^k;\ \mu^k \leq M1;\ \nu^k \geq 0\}$$

$$+ (1-\alpha)\sum_k \text{MAX}\{r^k\mu^k - \hat{y}^k\nu^k:$$

$$\mu^k A - \nu^k \leq c^k;\ \mu^k \leq M1;\ \nu^k \geq 0\}$$

$$= \alpha z(\bar{y}^1,\ldots,\bar{y}^K) + (1-\alpha)z(\hat{y}^1,\ldots,\hat{y}^K).$$

Therefore z is convex over Y. ■

Proposition 3.9 [52]

Let $\bar{y} = (\bar{y}^1,\ldots,\bar{y}^K) \in Y$ be any allocation and let $(\bar{\mu}^k,\bar{\nu}^k)$ denote the corresponding optimal solution to $z^k(\bar{y}^k)$ for $k = 1,\ldots,K$. Then $\pi = (-\bar{\nu}^1,\ldots,-\bar{\nu}^K)$ is a subgradient of z at $\bar{y}$.

Proof

Let $y = (y^1,\ldots,y^K) \in Y$ be any allocation and let (μ^k,ν^k) denote the corresponding optimal solution to $z^k(y^k)$ for $k = 1,\ldots,K$. Then:

$$z(y^1,\ldots,y^K) - z(\bar{y}^1,\ldots,\bar{y}^K) = \sum_k (r^k\mu^k - y^k\nu^k) - \sum_k (r^k\bar{\mu}^k - \bar{y}^k\bar{\nu}^k)$$

$$\geq \sum_k (r^k\mu^k - y^k\nu^k) - \sum_k (r^k\bar{\mu}^k - \bar{y}^k\bar{\nu}^k)$$

$$= \sum_k (-\bar{\nu}^k)(y^k - \bar{y}^k).$$

Hence $-\gamma_1$ is a subgradient of z at $\bar{y}$. ■

Recall that the subgradient optimization algorithm requires a technique for projecting a point onto the feasible region. We now explore the projection operation for this problem.

Let us denote the feasible region for RP by Ω. That is, $\Omega = \{(y^1,\ldots,y^K): \sum_k y^k = u; 0 \leq y^k \leq v^k (k=1,\ldots,K)\}$. Given an arbitrary allocation, $(\bar{y}^1,\ldots,\bar{y}^K)$, to project it onto Ω, we solve :

$$\text{MIN}\{||(y^1,\ldots,y^K)-(\bar{y}^1,\ldots,\bar{y}^K)||: y \in \Omega\}$$
$$= \text{MIN}\{(\sum_{kj}(y_j^k-\bar{y}_j^k)^2)^{1/2} : y \in \Omega\}.$$

Or, equivalently, we can solve:

$$\text{MIN}\{\sum_{kj}(y_j^k-\bar{y}_j^k)^2 : y \in \Omega\}.$$

Note that this problem decomposes on j. Hence, for each arc j, we solve:

$$\text{MIN}\{\sum_k (y_j^k-\bar{y}_j^k)^2 : \sum_k y_j^k = u_j;\ 0 \leq y_j^k \leq v_j^k ;\ (k=1,\ldots,K)\}.$$

We will denote the above projection problem by P. The following algorithm [52] is used to solve P for any arc, j.

ALGORITHM 3.3 PROJECTION ALGORITHM

Step 0 (Initialization)

If $u_j > \sum_k v_j^k$ or $u_j < 0$, terminate with no feasible solution. Otherwise set $l \leftarrow 1$; $r \leftarrow 2K$; $L \leftarrow \sum_k v_j^k$; $R \leftarrow 0$. Compute

the breakpoints, b_i $(i=1,\ldots,2K)$, as $\bar{y}_j^k$ and $\bar{y}_j^k - v_j^k$ $(k=1,\ldots,K)$.

Order the breakpoints so that $b_1 \leq b_2 \leq \ldots \leq b_{2K}$.

Step 1 (Test for Bracketing)

If $r-l=1$ go to step 4; otherwise, set $m \leftarrow [(l+r)/2]_I$ where $[K]_I$ is the greatest integer $\leq K$.

Step 2 (Computer New Value)

$$\text{Set } C \leftarrow \sum_k \text{MAX}\{\text{MIN}[\bar{y}_j^k - y_m,\ v_j^k],\ 0\}$$

Step 3 (Update)

If $C=c$ then set $\lambda \leftarrow y_m^*$ and go to step 5. If $C>c$ then set $l \leftarrow m$; $L \leftarrow C$; and go to step 1. If $C<c$ then set $r \leftarrow m$; $R \leftarrow C$; and go to step 1.

Step 4 (Interpolate)

$$\text{Set } \lambda^* \leftarrow b_l + [(b_r - b_l)(c-L)]/(R-L).$$

Step 5

Compute the feasible (projected) allocation, y_j^k, for $k=1,\ldots,K$ in this way:

$$y_j^k = \begin{cases} v_j^k, & \text{if } \lambda^* \leq \bar{y}_j^k - v_j^k \\ \bar{y}_j^k - \lambda^*, & \text{if } \bar{y}_j^k - v_j^k < \lambda^* \leq \bar{y}_j^k \\ 0, & \text{if } \lambda^* > \bar{y}_j^k \ . \end{cases}$$

Terminate with the feasible allocation for arc j, $(y_j^1,\ldots,y_j^K)$.

An upper bound algorithm using the subgradient procedure is now presented. Its convergence is a direct result of Proposition 3.4.

ALGORITHM 3.4 UPPER BOUND ALGORITHM

Step 0 (Initialization)

Let LB be any lower bound on the solution to MP. Choose a set of initial allocations, $y_0 = (y_0^1,\ldots,y_0^K)$ by setting $y_0^k \leftarrow P[(1/K)(u)]$ for $k = 1,\ldots,K$. Set $\lambda_0 \leftarrow 2$; $i \leftarrow 0$; $UB \leftarrow \infty$.

Step 1 (Find Subgradient)

Let (μ_i^k, π_i^k) solve $z^k(y_i^k)$ for $k = 1,\ldots,K$. Let $\eta_i \leftarrow (-\pi_i^1,\ldots,-\pi_i^K)$. Set $UB \leftarrow z^k(y_i^k)$. If $\eta_i = 0$, then terminate with $z(y_i)$ optimal.

Step 2 (Move to New Point)

Compute $s_i \leftarrow \lambda_i[z(y_i)-LB]/||\eta_i||^2$. Set $y_{i+1} \leftarrow P[y_i - s_i\eta_i]$. Set $\lambda_{i+1} \leftarrow \lambda_i/2$; $i \leftarrow i+1$. Return to step 1.

We now introduce a heuristic modification of the upper bound algorithm, which has produced better results on our test problems. Recall that $\eta = (-\pi^1,\ldots,-\pi^K)$ is a subgradient of z at $(y^1,\ldots,y^K)$. Then for each arc j, the vector

$$\eta(j) = (\eta_j e_j, \eta_{n+j} e_j, \ldots, \eta_{(k-1)n+j} e_j)$$

serves to isolate the components of η associated with the commodities flowing on arc j. For each such arc j we compute an individual step size at iteration i as

$$s_i(j) = \lambda_i[z(y_i^1,\ldots,y_i^K) - z^*]/||\eta_i(j)||^2$$

where z^* is approximated by LB.

Using this idea we now present our heuristic upper bound algorithm.

ALGORITHM 3.5 HEURISTIC UPPER BOUND ALGORITHM

Step 0 (Initialization)

Let LB be any lower bound on the solution to MP. Choose a set of initial allocations, $y_0 = (y_0^1,\ldots,y_0^K)$ by setting $y_0^k \leftarrow P[(1/K)(u)]$ for $k = 1,\ldots,K$. Set $\lambda_0 \leftarrow 2$; $i \leftarrow 0$; $UB \leftarrow \infty$.

Step 1 (Find Subgradient)

Let (ν_i^k, v_i^k) solve $z^k(y_i^k)$ for $k = 1,\ldots,K$. Let $\pi_i = (-v_i^1,\ldots,-v_i^K)$. Set $UB \leftarrow \sum_k z^k(y_i^k)$. If $\pi_i = 0$, then terminate with $z(y_i)$ optimal.

Step 2 (Move to New Point)

Compute $s_i(j) \leftarrow \lambda_i[z(y_i^1,\ldots,y_i^K)-LB]/||\pi_i(j)||^2$ for each arc j. Set $\hat{S} \leftarrow \text{diag}(s_i(1),\ldots,s_i(n))$. Set

$$S \leftarrow \begin{bmatrix} \hat{S} & & & \\ & \hat{S} & & \\ & & \ddots & \\ & & & \hat{S} \end{bmatrix}.$$

Set $(y_{i+1}^1,\ldots,y_{i+1}^K) \leftarrow P[(y_i^1,\ldots,y_i^K)-S\pi_i]$. Set $\lambda_{i+1} \leftarrow \lambda_i/2$; $i \leftarrow i+1$.

Go to step 1.

3.4 The Algorithm

In this section we present the composite algorithm for solving MP. This procedure involves partially solving DP for successively better lower bounds and partially solving RP for successively better upper bounds on the optimal objective function value. The algorithm terminates whenever (a) the solution to DP can be shown to be an exact optimum; (b) the solution to RP can be shown to be an exact optimum; or (c) the greatest lower bound and the least upper bound generated are within a prescribed tolerance, ε. In case (c), the best solution to RP is presented as a guaranteed ε-optimal solution.

ALGORITHM 3.6 COMPLETE ALGORITHM

Step 0 (Initialization)

Let ε←termination tolerance ($0<\varepsilon<1$); NOLB←number of lower bound iterations to perform on each pass; NOUB←number of upper bound iterations to perform on each pass; LB←$-\infty$; UB←∞.

Step 1 (Lower Bound)

Perform NOLB iterations of the lower bound algorithm (Algorithm 3.2). Let LB denote the best lower bound attained so far. If Algorithm 3.2 terminates in step 1 with an exact optimum, terminate with that solution optimal for MP.

Step 2 (Upper Bound)

Perform NOUB iterations of an upper bound algorithm (Algorithm 3.4 or 3.5). Let UB denote the best upper bound attained so far. If Algorithm 3.4 terminates in step 1 with an exact optimum, terminate with that solution optimal for MP.

Step 3 (Check for Termination)

If $\varepsilon(UB) \leq LB$ then terminate with UB a guaranteed ε-optimum; otherwise, go to step 1.

In this algorithm the best solutions for the lower bound and upper bound problems at each pass are retained and used as starting solutions for the respective problems on the next pass. The details of our implementation are presented in Chapter 4.

CHAPTER IV

COMPUTATIONAL EXPERIMENTATION

This chapter provides descriptions of our computer implementation of Algorithm 3.6 and of the test problems used. Our code, EVAC, uses MODFLO [1] to solve the single commodity minimum cost network flow subproblems which arise in Algorithm 3.2 and in Algorithm 3.5. MODFLO is a set of routines which may be used to solve a network flow problem or to reoptimize a previously solved problem after changes are made in some of the data. MODFLO, which is based on NETFLO [52], allows the user to change bounds, costs, and/or requirements and then reoptimize from a basis which was optimal for the original problem.

We tested EVAC on 22 randomly generated multicommodity network flow problems and on one test problem which was specially structured to be solved by EVAC. The test problems ranged in size from 22 to 754 nodes and from 53 to 1,102 arcs with from 0 to 599 linking constraints and from 3 to 20 commodities. The equivalent LP sizes are between 232 and 8,904 rows and between 470 and 12,111 columns. The 22 randomly generated problems were created using MNETGN [5], a multicommodity network problem generator. The problems were solved by EVAC and by MCNF [51], a multicommodity network flow code which uses a primal partitioning algorithm. Solution times are compared and conclusions are drawn concerning the relative effectiveness of the techniques.

4.1 Description of the Computer Programs

In this section we present a description of MCNF and EVAC, the two computer codes used in our experimentation. Both programs are written in standard FORTRAN and have been tailored to neither our equipment nor our FORTRAN compiler.

4.1.1 MCNF

MCNF was developed by Jeff Kennington at Southern Methodist University, Dallas, TX. It is an incore multicommodity network flow problem solver which uses the modification of the revised simplex method known as the primal partitioning algorithm [36]. In this algorithm the basis inverse is maintained as a set of rooted spanning trees (one for each commodity) and a working basis inverse is maintained in product form. The working basis inverse has dimension equal to the number of binding linking constraints corresponding to the current basis. The initial basis is created using a multicommodity variation of the routine used in NETFLO. A partial pricing scheme is used; the pricing tolerance is 1.E-6 and the pivot tolerance is 1.E-8.

4.1.2 EVAC

EVAC is our implementation of Algorithm 3.6 for solving the multicommodity network flow problem. Note that Algorithm 3.6 alternates between generating lower bounds using Algorithm 3.2 and generating upper bounds using Algorithm 3.5. Since both the lower bound problem (DP) and the upper bound problem (RP) decompose on commodities, EVAC maintains only the information concerning the current commodity in main memory. The problem data and most recent bases for all the other commodities are

kept on peripheral storage. At the user's option EVAC stores in main memory as much of the current set of allocations, $(y_i^1,\ldots,y_i^k)$ and current dual variables $(-v_i^1,\ldots,-v_i^k)$ as desired. All our test problems (with the exception of Problem 23) were solved with all the allocations and dual variables in core.

Both the lower bound routine and the upper bound routine use MODFLO as the optimizer for the single commodity subproblems. MODFLO uses the same partial pricing scheme as NETFLO and drives the flow on artificial arcs to zero using the Big-M method. The Big-M value that was used for our test problems, except as noted in Table 4.1, was 7 times the largest unit cost in the given problem. At subsequent iterations, initial bases for each commodity are just the optimal bases for the previous set of Lagrange multipliers. A basis for the upper-bound problem is generated by constructing a feasible basis from the previous optimal basis using the rules described in [1].

In practice we did not update the multipliers for the step sizes (α_i in Algorithm 3.2 and λ_i in Algorithm 3.5) at every iteration, but only when the improvement in the objective function was too small. As Algorithm 3.2 requires a finitie upper bound (for calculation of the step size in step 2) we used an initial value of UB ← 1.1*LB. Thereafter for UB we used the best upper bound generated so far. The parameters and tolerance used in all our testing were these:

ε = .90

NOLB = 5

NOUB = 5

Pricing Tolerance = 1.E-2

4.2 Description of the Test Problems

The multicommodity network problem generator, MNETGN, was used to create 22 random test problems. We modified the MNETGN output so that every arc appeared in every commodity's subproblem by adding arcs with upper bounds of zero where necessary. The test problem ranged in size from 22 to 754 nodes and from 53 to 1,102 arcs with from 0 to 599 linking constraints and from 3 to 20 commodities. The equivalent LP sizes are between 232 and 8,904 rows and between 470 and 12,111 columns. The number of linking constraints corresponds to a wide variety of problems from pure network problems (no linking constraints) to problems in which over 75% of the arcs are included in linking constraints.

Problem 15 was provided by Lt. Col. Dennis McLain, the Assistant Director of Operations Research at the Military Airlift Command located at Scott Air Force Base.

4.3 Summary of Computational Results

All the testing (except for Problems 15, 21, and 23) was done on a CDC 6600 at Southern Methodist University, using the FTN compiler with the optimization feature enabled. Except for Problems 7 and 23, a guaranteed ε-optimum was obtained for each problem with $\varepsilon \geq 90\%$. Problem 7 experienced convergence difficulties when run using EVAC. Problem 8 was created from Problem 7 by increasing the linking constraint bounds by 10%. As indicated in Table 4.1, this slight modification enabled EVAC to solve the problem easily. We limited the number of lower bound iterations and upper bounds iterations to 100.

even though Problem 7 had not achieved 90% optimality by that point. Because of this the solution times for Problem 7 are given in Table 4.1 but are not included in the summary data.

Problem 23 was created to allow us to test EVAC on a relatively large problem. This problem (with 8,904 LP rows and 12,111 LP columns) was too large for MCNF to solve in the available memory, so we were not able to compare solution times for the two codes on this problem. In addition, due to the memory limitations on the CDC 6600, we were forced to use a CDC 205 to test Problem 23. For this reason the times for Problem 23 are included in Tables 4.1 and 4.2, but are not included in the totals and summary information. Since the testing on the CDC 205 involved a real-dollar expense, we were satisfied to stop when a 75% optimum was attained. The test runs for Problems 15 and 21 were made on a CDC Cyber 73. But since both the EVAC and MCNF runs for these problems were made on the Cyber 73, the totals and summary data include the times for Problems 15 and 21.

Details of the test problems are given in Table 4.1. The times are in CPU seconds and exclude the time required to input the problem data and print the solution reports. Table 4.1 also presents a comparison of the times required for MCNF and EVAC to solve each problem. In order to present a meaningful comparison of the solution times for MCNF and EVAC, we also present the solution times for EVAC exclusive of the extra I/O required to maintain the costs, bounds, and old bases for the subproblems on peripheral storage. Since MCNF maintains all this information in main memory, this seems to be the most reasonable way of comparing timing statistics. The column titled "Guaranteed % Optimal" gives the best lower bound generated by EVAC as a percent of the best

upper bound generated by EVAC. The column titled "Actual % Optimal" presents the actual optimal objective (as obtained by MCNF) as a percent of the best upper bound generated by EVAC.

Table 4.2 provides the details of the times required by EVAC to perform various steps of the algorithm. The column titled "% of Time in Other" for the lower bound computations shows the time required for such activities as computing the Lagrange multipliers, updating the unit costs to reflect these changes, computing the resulting dual variables, and various bookkeeping activities. The corresponding column for upper bound computations reflects such activities as calculating the dual variables, testing the termination criteria, and various other short computations.

Table 4.3 summarizes the time comparisons graphically. The problems are grouped by number of commodities, as they are in Tables 4.1 and 4.2.

4.4 Analysis of Results

It seems clear from Tables 4.1 and 4.3 that EVAC severely dominates MCNF whenever the number of commodities is small. This is due to the fact that, for EVAC, quite a bit of additional overhead is involved in alternating between commodities. This overhead is not just a result of I/O, although that is a great deal of it, but is also due to the set-up time required for activities such as constructing a new feasible basis from an old basis and calculating the resulting dual variables. MCNF, on the other hand, is primarily driven by the number of binding linking constraints in the optimal solution. This is because MCNF seeks an exact optimum.

Letting T(EVAC) denote the average time required to EVAC (exclusive of I/O), and T(MCNF) denote the average time required to MCNF, we can express the following relationships:

For the 3-commodity test problems,

$$T(EVAC) = .354 * T(MCNF).$$

For the 4-commodity test problems,

$$T(EVAC) = .469 * T(MCNF).$$

For the 5-commodity test problems,

$$T(EVAC) = .666 * T(MCNF).$$

And for the test problems with 6 or more commodities,

$$T(EVAC) = .975 * T(MCNF).$$

It should also be noted that EVAC is capable of solving larger problems than MCNF. This is due to the fact that EVAC stores only one copy of the network defining data in main memory, where MCNF requires one copy for each commodity. Also, EVAC maintains in main memory the current basis, cost and bound data for only one commodity at a time. Thus, for a K-commodity problem, EVAC uses on the order of 1/K the main memory required by MCNF.

Note that the entries in the "Guaranteed % Optimal" and "Actual % Optimal" columns of Table 4.1 are quite close. This indicates that the sequence of lower bounds converged to values very near optimality. In addition, from Table 4.2, we see that the lower bound iterations are typically less time consuming than the upper bound iterations.

It is worth observing that EVAC was designed for very large problems which would never be solved to optimality. Even if a problem does not converge to within the requested tolerance in a prescribed number of iterations, EVAC always provides a feasible solution which is

a guaranteed ε-optimum for some $\varepsilon > 0$. In contrast, MCNF provides only an upper bound on the optimum objective value, with no indication of how close it is to optimality until an exact optimum is actually attained.

We conclude that EVAC works extremely well in obtaining a guaranteed ε-optimum for the multicommodity network flow problem. While it is not as "robust" as the simplex-based MCNF, it is a good choice for the class of problems for which it was developed, the very large casualty evacuation models.

TABLE 4.1
DESCRIPTION OF THE TEST PROBLEMS AND SUMMARY COMPARISON OF SOLUTION TIMES FOR EVAC AND MCNF

PROBLEM	COMMOD-ITIES	NODES	ARCS	L.P. ROWS	L.P. COLUMNS	LINKING CONSTRAINTS	BINDING LINKING CONSTRAINTS	% IN L.B. ROUTINE (EVAC)	% IN U.B. ROUTINE (EVAC)	TOTAL EVAC TIME	TOTAL EVAC TIME (EXCLUDING I/O)	TOTAL MCNF TIME	GUARANTEED % OPTIMAL	ACTUAL % OPTIMAL
1	3	253	657	1,174	2,352	412	6	81%	19%	7.97	4.22	19.41	94.3%	95.2%
2	3	142	480	721	1,678	292	11	73%	27%	5.81	3.31	21.52	91.5%	92.8%
3	3	108	336	581	1,214	254	27	22%	78%	19.46	17.12	28.71	90.0%	91.6%
TOTAL FOR 3-COMMODITY TEST PROBLEMS											24.65	69.64		
4	4	104	314	676	1,494	256	5	45%	55%	11.70	8.42	18.21	94.6%	94.9%
5	4	196	524	1,124	2,393	336	6	83%	17%	9.63	4.53	22.11	93.4%	94.3%
6	4	116	377	741	1,724	273	8	43%	57%	13.60	9.79	18.70	93.2%	93.3%
7 (*)	4	105	338	636	1,485	212	16	--	--	--	--	24.51	--	--
8	4	105	338	636	1,485	212	13	32%	68%	19.90	16.12	23.70	90.0%	91.0%
9	4	107	358	636	1,554	204	15	36%	64%	17.26	13.46	28.78	90.8%	92.4%
TOTAL FOR 4-COMMODITY TEST PROBLEMS											52.32	111.50		
10	5	101	259	510	1,214	0	0	100%	0%	5.29	2.17	11.65	100%	100%
11 (**)	5	101	246	511	1,154	1	0	100%	0%	5.20	2.18	10.99	100%	100%
12 (**)	5	104	245	584	1,212	59	7	58%	42%	9.44	6.14	14.71	93.3%	93.3%
13	5	92	295	680	1,622	215	10	32%	68%	26.78	21.16	23.00	90.6%	91.0%
14	5	85	268	626	1,439	196	18	27%	73%	31.79	26.39	26.79	90.2%	91.2%
TOTAL FOR 5-COMMODITY TEST PROBLEMS											58.04	87.14		
15	6	48	131	232	470	84	5	89%	11%	2.51	1.13	4.54	93.1%	93.3%
16	20	23	53	480	964	0	0	100%	0%	14.04	3.65	6.83	100%	100%
17	20	22	59	461	1,046	1	0	100%	0%	15.27	3.51	6.50	100%	100%
18	10	48	126	589	1,296	99	2	74%	26%	13.81	6.35	12.04	95.7%	97.0%
19	20	22	53	475	980	15	2	86%	14%	16.84	5.73	6.94	92.8%	93.3%
20	12	45	121	633	1,367	81	4	49%	51%	29.10	18.33	14.40	91.9%	91.9%
21	6	71	238	609	1,524	177	9	31%	69%	50.55	40.44	36.33	92.5%	92.7%
22 (**)	6	76	219	619	1,361	157	26	22%	78%	50.04	43.03	37.92	90.8%	91.1%
23 (*) (**)	11	754	1,102	8,904	12,111	599	--	72%	28%	327.18	--	--	75.3%	--
TOTAL FOR $\geq$ 6 COMMODITY TEST PROBLEMS											122.35	125.50		
TOTAL FOR ALL TEST PROBLEMS											257.36	393.78		

(*) PROBLEM NOT INCLUDED IN SUMMARY INFORMATION
(**) PROBLEM REQUIRED A BIG-M VALUE > 7 * LARGEST UNIT COST

TABLE 4.2
DETAILED TIMING STATISTICS FOR EVAC RUNS

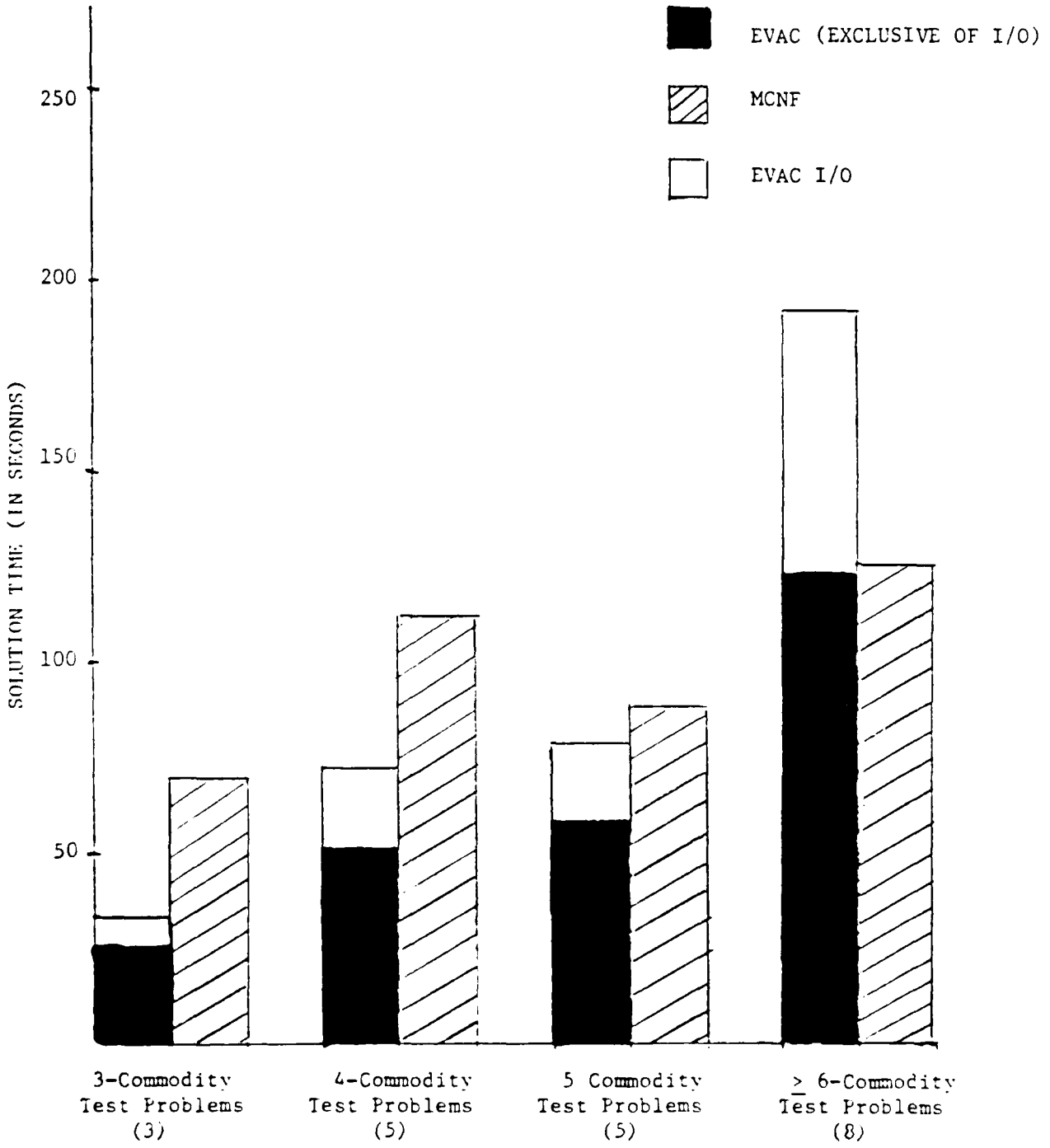
		LOWER BOUND ROUTINE						UPPER BOUND ROUTINE							
PROBLEM	COMMOD-ITIES	% TIME IN I/O	% TIME IN OPTIMIZER	% TIME FINDING SUBGRAD.	% TIME IN OTHER	% TIME IN L.B. ROUTINE	# L.B. ITERA-TIONS	% TIME IN I/O	% TIME IN OPTIMIZER	% TIME FINDING SUBGRAD.	% TIME CREATING FEAS.BASIS	% TIME IN PRO-JECTION	% TIME IN OTHER	% TIME IN U.B. ROUTINE	# U.B. ITERA-TIONS
1	3	46%	16%	0%	19%	81%	5	1%	9%	2%	3%	2%	2%	19%	2
2	3	42%	15%	0%	16%	73%	5	1%	16%	2%	5%	2%	1%	27%	2
3	3	10%	6%	0%	6%	22%	20	2%	44%	4%	18%	9%	3%	78%	20
AVG FOR 3	3	33%	12%	0%	14%	59%	10	1%	23%	3%	9%	4%	2%	41%	8
4	4	26%	9%	0%	10%	45%	10	2%	28%	4%	11%	9%	1%	55%	9
5	4	53%	11%	0%	19%	83%	5	1%	8%	1%	3%	2%	2%	17%	2
6	4	26%	7%	0%	10%	43%	10	2%	31%	4%	9%	9%	2%	57%	10
7 (*)	4	--	--	-	--	--	--	-	--	-	-	-	-	--	--
8	4	17%	8%	0%	7%	32%	20	2%	39%	4%	14%	9%	0%	68%	17
9	4	20%	8%	0%	8%	36%	15	2%	38%	3%	13%	7%	1%	64%	12
AVG FOR 4	4	28%	9%	0%	11%	48%	12	2%	29%	3%	10%	7%	1%	52%	10
10	5	59%	18%	0%	23%	100%	1	0%	0%	0%	0%	0%	0%	0%	0
11 (**)	5	58%	19%	0%	23%	100%	1	0%	0%	0%	0%	0%	0%	0%	0
12 (**)	5	33%	10%	0%	15%	58%	5	2%	22%	1%	14%	2%	1%	42%	5
13	5	18%	6%	0%	8%	32%	25	3%	35%	5%	13%	12%	0%	68%	23
14	5	14%	6%	0%	7%	27%	35	3%	36%	5%	16%	13%	0%	73%	31
AVG FOR 5	5	36%	12%	0%	15%	63%	13	2%	19%	2%	9%	5%	0%	37%	12
15	6	54%	14%	0%	21%	89%	5	1%	4%	1%	2%	0%	3%	11%	1
16	20	74%	3%	0%	23%	100%	1	0%	0%	0%	0%	0%	0%	0%	0
17	20	77%	2%	0%	21%	100%	1	0%	0%	0%	0%	0%	0%	0%	0
18	10	53%	5%	0%	16%	74%	10	2%	8%	2%	6%	7%	1%	26%	6
19	20	64%	3%	0%	19%	86%	5	2%	4%	2%	4%	2%	1%	14%	5
20	12	33%	4%	0%	12%	49%	20	3%	18%	3%	14%	12%	1%	51%	18
21	6	18%	6%	0%	7%	31%	25	2%	31%	2%	14%	15%	1%	69%	22
22 (**)	6	11%	6%	0%	5%	22%	50	3%	33%	3%	22%	14%	2%	78%	50
23 (*)(**)	11	8%	4%	0%	5%	17%	65	0%	18%	0%	9%	52%	1%	83%	65
AVG For ≥ 6	17	48%	5%	0%	16%	69%	15	2%	12%	2%	8%	6%	1%	31%	13

(*) PROBLEM NOT INCLUDED IN SUMMARY INFORMATION

(**) PROBLEM REQUIRED A BIG-M- VALUE > 7*LARGEST UNIT COST

TABLE 4.3

GRAPHICAL COMPARISON OF EVAC AND MCNF SOLUTION TIMES



CHAPTER V

SUMMARY AND CONCLUSIONS

This chapter presents a summary of the results reported in Chapter IV and shares conclusions regarding the relative effectiveness of our technique. It also includes ideas for further investigation in the area.

5.1 Summary and Conclusions

Algorithm 3.6 describes our technique for finding an ε-optimal solution for the multicommodity network flow problem. Our technique differs from other approaches to the problem in that, rather than solving the multicommodity problem directly, we compute sequences of lower and upper bounds on the optimal objective function value, terminating when the bounds are within a prescribed tolerance. Both the lower and upper bound algorithms use a subgradient optimization technique and both decompose on commodities so that only a single commodity minimum cost network flow optimizer is required. At each iteration of the lower bound routine (Algorithm 3.2), an initial basis is generated from the previous optimal basis by modifying the costs to correspond to the new Lagrange multipliers, and updating the dual variables. At each iteration of the upper bound routine (Algorithm 3.5), an initial basis is constructed from the previous optimal basis

using the rules described in [1] to restore feasibility (if necessary) after changing the bounds to correspond to the new allocations.

The subgradients for the lower bounds are computed to be the sum of the flows on the mutually constrained arcs minus the associated mutual arc capacities. For the upper bounds, subgradients are computed using the dual variables obtained when solving the single commodity network problems.

Our computational work included solving each one of 23 problems twice; once using MCNF, a primal partitioning code, and once using EVAC, our implementation of Algorithm 3.6. On the average EVAC required only 65% of the time required by MCNF (ignoring I/O). EVAC's performance was far superior on the problems with fewer commodities and was not as impressive on the problems involving many commodities. In addition EVAC required on the order 1/K the amount of main memory as MCNF for a K-commodity problem.

5.2 Areas for Future Investigation

Algorithm 3.6 involves two more or less independent processes. That is, there is no reason why the lower bound generator (Algorithm 3.2) and the upper bound generator (Algorithm 3.5) could not proceed independently, stopping now and then to exchange their best bounds and test for optimality. Hence it appears that this procedure is well-suited to exploit the benefits of a parallel processing environment. In addition to the partitioning of the technique into two separate procedures, within each of these procedures the decomposition by commodities could take advantage of a parallel

processing scheme as well. It would seem reasonable to expect such a scheme to speed up the execution time considerably, especially when solving a very large problem.

There is also room for additional experimentation with the step sizes, specifically with the multipliers on the step sizes. Perhaps a scheme in which the multipliers were allowed to be reset to their starting values a finite number of times would speed up convergence. One might reset these multipliers whenever the improvement in the sequence of upper (lower) bounds fell below some tolerance. This would have the effect of restarting the algorithm at that point, but with a far better "starting solution".

In addition this problem has a multiperiod structure. Since the network is replicated for 60 one day time periods, it might be advantageous to exploit this structure using a forward simplex approach.

LIST OF REFERENCES

1. Ali, A., Allen, E., Barr, R., and Kennington, J., "Reoptimization Procedures for Bounded Variable Primal Simplex Network Algorithms", to appear in European Journal of Operations Research.

2. Ali, I., Barnett, D., Farhangian, K., Kennington, J., McCarl, B., Patty, B., Shetty, B., and Wong, P., "Multicommodity Network Problems: Applications and Computations," IIE Transactions, 16, 2, 127-134 (1984).

3. Ali, A. I., Helgason, R. V., Kennington, J. L., and Lall, H., "Primal-Simplex Network Codes: State-of-the-Art Implementation Technology," Networks, 8, 315-339 (1978).

4. Ali, A. I., Helgason, R. V., Kennington, J. L., and Lall, H., "Computational Comparison among Three Multicommodity Network Flow Algorithms," Operations Research, 23, 995-1000 (1980).

5. Ali, A. and Kennington, J., "MNETGN Program Documentation", Technical Report IEOR 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, TX, (1977).

6. Ali, A. I., and Kennington, J. L., "Network Structure in Linear Programs: A Computational Study," Technical Report No. 83-OR-1, Department of Operations Research, Southern Methodist University, Dallas, TX (1983).

7. Ali, A., and Kennington, J., "The Asymmetric M-Travelling Salesman Problem: A Duality Based Branch-And-Bound Algorithm," to appear in Discrete Applied Mathematics.

8. Assad, A. A., "Multicommodity Network Flows -Computational Experience," Working Paper OR-058-76, Operatons Research Center, Massachusetts Institute of Technology, Cambridge, MA, (1976).

9. Barr, R. S., Glover, F., and Klingman, D., "The Alternating Basis Algorithm for Assignment Problems," Mathematical Programming, 13, 1, 1-13 (1977).

10. Barr, R. S., Glover, F., and Klingman, D., "Enchancements of Spanning Tree Labelling Procedures for Network Optimization," INFOR, 17, 1, 16-34 (1979).

11. Bazaraa, M., and Goode, J., "The Travelling Salesman Problems: A Duality Approach," Mathematical Programming, 13, 221-237 (1977).

12. Bazarra, M. and Shetty, C., Nonlinear Programming: Theory and Algorithms, John Wiley and Sons, New York, NY, (1979).

13. Bradley, G. H., Brown, G. G., and Graves, G. W., "Design and Implementation of Large-Scale Primal Transshipment Algorithms," Management Science, 24, 1, 1-34 (1977).

14. Charnes, A., and Cooper, W. W., Management Models and Industrial Applications of Linear Programming, Vols. 1 and 2, John Wiley and Sons, New York, NY, (1961).

15. Chen, H., and DeWald, C. G., "A Generalized Chain Labeling Algorithm for Solving Multicommodity Flow Problems," Computers and Operations Research, 1, 437-465 (1974).

16. Cremeans, J. E., Smith, R. A., and Tyndall, G. R., "Optimal Multicommodity Network Flows with Resource Allocation," Naval Research Logistics Quarterly, 17, 269-280 (1970).

17. Dantzig, G. B., "Application of the Simplex Method to a Transportation Problem," in T. C. Koopmans, Ed., Activity Analysis of Production and Allocation, John Wiley and Sons, New York, NY, (1951).

18. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, NJ (1963).

19. Dantzig, G. B., and Wolfe, P., "Decomposition Principle for Linear Programs," Operations Research 8, 101-111 (1960).

20. Ford, L. R., and Fulkerson, D. R., "Maximal Flow through a Network," Canadian Journal of Mathematics, 8, 3, 399-404 (1956).

21. Ford, L. R., and Fulkeson, D. R., "A Suggested Computation for Maximal Multicommodity Network Flow," Management Science, 5, 97-101 (1958).

22. Ford, L. R., and Fulkerson, D. R., Flows in Networks, Princeton University Press, Princeton, NJ, (1962).

23. Fulkerson, D. R., "An Out-of-Killer Method for Minimal-Cost Flow Problems," Journal of the Society of Industrial and Applied Mathematics, 9, 1, 18-27 (1961).

24. Glover, F., Glover, R., and Martinson, F., "The U.S. Bureau of Land Management's New NETFORM Vegetation Allocation System," Technical Report of the Division of Information Science Research, University of Colorado, Boulder, CO (1982).

25. Glover, F., Hultz, J., and Klingman, D., "Improved Computer-Based Planning Techniques," Research Report CCS 283, Center for Cybernetic Studies, The University of Texas, Austin, TX, (1977).

26. Glover, F., Hultz, J., and Klingman, D., "Network Versus Linear Programming Algorithms and Implementations," CCS 306, Center for Cybernetic Studies, The University of Texas, Austin, TX, (1977).

27. Glover, F., Karney, D., and Klingman, D., "Implementation and Computational Comparisons of Primal, Dual, and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," Networks, 4,3, 191-212 (1974).

28. GLover, F., Karney,D., Klingman, D., and Napier, A., "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Management Science, 20, 5, 793-813 (1974).

29. Glover, F., and Klingman, D., "New Advances in the Solution of Large-Scale Network and Network-Related Problems," Technical Report CCS 177, Center for Cybernetic Studies, The University of Texas, Austin, Tx, (1974).

30. Glover, F., and Klingman, D., "New Advances in the Solution of Large-Scale Network and Network-Related Problems," CCS 238, Center for Cybernetic Studies, The University of Texas, Austin, TX, (1975).

31. Glover, F., and Klingman, D., "Some Recent Practical Misconceptions about the State-of-the-Art of Network Algorithms," Operations Research, 2, 370-379 (1978).

32. Glover, F., Klingman, D., and Stutz, J., "Augmented Threaded Index Method for Network Optimization," INFOR, 12, 3, 293-298 (1974).

33. Graves, G. W., and McBride, R. D., "The Factorization Approach to Large-Scale Linear Programming," Mathematical Programming, 10, 1, 91-110 (1976).

34. Grigoriadis, M.D., and White, W. W., "A Partitioning Algorithm for the Multicommodity Network Flow Problem," Mathematical Programming, 3, 157-177 (1972).

35. Hartman, J. K., and Lasdon, L. S., "A Generalized Upper Bounding Method for Doubly Coupled Linear Programs," Naval Research Logistics Quarterly, 17, 4, 411-429 (1970).

36. Hartman, J., and Lasdon, L., "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems", Networks, 1, 333-354, (1972).

37. Held, M., and Karp, T., "The Travelling Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming, 1, 6-25 (1971).

38. Held, M., Wolfe, P., and Crowder, H., "Validation of Subgradient Optimization", Mathematical Programming, 6, 66-68, (1974).

39. Helgason, R., "A Lagrangian Relaxation Approach to the Generalized Fixed Charge Multicommodity Minimum Cost Network Flow Problem," unpublished dissertation, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, TX, (1980).

40. Helgason, R. V., and Kennington, J. L., "A Product Form Representation of the Inverse of a Multicommodity Cycle Matrix," Networks, 7, 297-322 (1977).

41. Helgason, R. V., and Kennington, J. L., "An Efficient Procedure for Implementing a Dual-Simplex Network Flow Algorithm," AIIE Transactions, 9, 1, 63-68 (1977).

42. Hitchcock, F. L., "The Distribution of a Product from Several Sources to Numerous Localities," Journal of Mathematics and Physics, 20, 224-230 (1941).

43. Jarvis, J. J., "On the Equivalence Between the Node-Arc and Arc-Chain Formulation for the Multicommodity Maximal Flow Problem," Naval Research Logistics Quarterly, 15, 525-529 (1969).

44. Jarvis, J. J., and Keith, P. D., "Multicommodity Flows with Upper and Lower Bounds," Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, (1974).

45. Jarvis, J. J., and Martinez, O. M., "A Sensitivity Analysis of Multicommodity Network Flows," Transportation Science, 11, 4, 299-306 (1977).

46. Jewell, W. S., "A Primal-Dual Multicommodity Flow Algorithm," ORC 66-24, Operations Research Center, University of California, Berkeley, CA, (1966).

47. Johnson, E. L., "Programming in Networks and Graphs," Technical Report ORC 65-1, Operations Research Center, University of California at Berkeley (1965).

48. Kantorovich, L.V., "Mathematical Methods in the Organization and Planning of Production," Publication House of the Leningrad State University, 1939. 68pp. Translated in Management Science, 6, 366-422 (1960).

49. Karney, D., and Klingman, D., "Implementation and Computational Study on an In-core, Out-of-core Primal Network Code," Operations Research, 24, 1056-1077 (1976).

50. Kennington, J. L., "Solving Multicommodity Transportation Problems Using a Primal Partitioning Simplex Technique," Naval Research Logistics Quarterly, 24, 2, 309-325 (1977).

51. Kennington, J., "A Primal Partitioning Code for Solving Multicommodity Network Flow Problems", Technical Report No. 79008, Department of Operations Research, Southern Methodist University, Dallas, TX, (1979).

52. Kennington, J., and Helgason, R., Algorithms for Network Programming, John Wiley & Sons, New York, NY, (1980).

53. Kennington, J. L., and Shalaby, M., "An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems," Management Science, 23, 9, 994-1004 (1977).

54. Koopmans, T. C., and Reiter, S., "A Model of Transportation," in T. C. Koopmans, Ed., Activity Analysis of Production and Allocation, John Wiley and Sons, New York, NY, (1951).

55. Kuhn, H. W., "The Hungarian Method for the Assignment Problem", Naval Research Logistics Quarterly, 2, 83-97 (1955).

56. Maier, S. F., "A Compact Inverse Scheme Applied to a Multi-commodity Network with Resource Constraints," in R. Cottle and J. Krarup, Eds., Optimization Methods for Resource Allocation, The English University Press, London, England (1974).

57. Mulvey, J. M., "Pivot Strategies for Primal-Simplex Network Codes," Journal of the Association for Computing Machinery, 25, 2, 266-270 (1978).

58. Mulvey, J., "Testing of a Large-scale Network Optimization Program," Mathematical Programming, 15, 291-314 (1978).

59. Orden, A., "The Transshipment Problem", Management Science, 2, 2, 276-285 (1956).

60. Robacker, J. T., "Notes on Linear Programming: Part XXXVII, Concerning Multicommodity Networks," Memo RM-1799, The Rand Corporation, Santa Monica, CA, (1956).

61. Saigal, R., "Multicommodity Flows in Directed Networks," ORC 67-38, Operations Research Center, University of California, Berkeley, CA, (1967).

62. Shor, N., "On the Structure of Algorithms for the Numerical Solution of Optimal Planning and Design Problems," unpublished dissertation, Cybernetics Institute, Academy of Sciences, U.S.S.R. (1964).

63. Srinivasan V., and Thompson, G. L., "Accelerated Algorithms for Labelling and Relabeling of Trees, with Applications to Distribution Problems," Journal of the Association for Computing Machinery, 19, 4, 712-726 (1972).

64. Srinivasan, V., and Thompson, G. L., "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," Journal of the Association for Computing Machinery, 20, 194-213 (1973).

65. Swoveland, C., "Decomposition Algorithms for the Multicommodity Distribution Problem," Working Paper 184, Western Management Science Institute, University of California, Los Angeles, CA, (1971).

66. Swoveland, C., "A Two-Stage Decomposition Algorithm for a Generalized Muticommodity Flow Problem," INFOR, 11, 232-244 (1973).

67. Tomlin, J. A., "Mathematical Programming Models for Traffic Network Problems," unpublished dissertation, Department of Mathematics, University of Adelaide, Australia (1967).

68. Weigel, H. S., and Cremeans, J. E., "The Multicommodity Network Flow Model Revised to Include Vehicle per Time Period and Mode Constraints," Naval Research Logistics Quarterly, 19, 77-89 (1972).

69. Wollmer, R. D., "Multicommodity Networks with Resource Constraints: The Generalized Multicommodity Flow Problems," Networks, 1, 245-263 (1972).

CHAPTER 2

Networks with Side Constraints: An LU Factorization Update

Richard S. Barr, Keyvan Farhangian, Jeffery L. Kennington

An important class of mathematical programming models which are frequently used in logistics studies is the model of a network problem having additional linear constraints. A specialization of the primal simplex algorithm which exploits the network structure can be applied to this problem class. This specialization maintains the basis as a rooted spanning tree and a general matrix called the working basis. This paper presents the algorithms which may be used to maintain the inverse of this working basis as an LU factorization, which is the industry standard for general linear programming software. Our specialized code exploits not only the network structure but also the sparsity characteristics of the working basis. Computational experimentation indicates that our LU implementation results in a 50 percent savings in the non-zero elements in the eta file, and our computer codes are approximately twice as fast as MINOS and XMP on a set of randomly generated multicommodity network flow problems.

ACKNOWLEDGEMENT

This research was supported in part by the Department of Defense under Contract Number MDA903-82-C-0440 and the Air Force Office of Scientific Research under Contract Number AFOSR 83-0278.

Good software for solving linear programming models is one of the most important tools available to the logistics engineer. For logistics studies, these linear programs frequently involve a very large network of nodes and arcs, which may be duplicated by time period. For example, nodes may represent given cities at a particular point in time while arcs represent roads, railways, and legs of flights connecting these cities. Some nodes are designated as supply nodes, others demand nodes, while some may simply represent points of transshipment. The mathematical model characterizes a solution such that the supply is shipped to the demand nodes at least cost while not violating either the upper or lower bounds on the flow over an arc.

If the main structure of a logistics problem can be captured in a network model, then the size of solvable problems becomes enormous. Hence, more realistic situations can be modelled that would otherwise lie outside the domain of general linear programming techniques. For example, one current logistics planning model involves 200 nodes and (365 days/yr) (30 years) = 10,950 time periods to give over 2,000,000 constraints. Network problems having 20,000 constraints and 20,000,000 variables are solved routinely at the U. S. Treasury Department.

Unfortunately, the pure network structure may require simplification of the problem to the point that key policy restrictions must be omitted. The work presented in this study builds upon existing large-scale network solution technology to allow for the inclusion of arbitrary additional constraints. Typical constraints include capacities on vehicles carrying different types of goods, restrictions on the total number of vehicles available for assignment, and budget restrictions. The addition of even a few non-network constraints can greatly enhance the realism and usability of these models. Our approach exploits—to as great an extent as possible—the traditional network portion of the problem while simultaneously enforcing any additional restrictions imposed by the practitioner.

For general linear programming systems, the most important component is the algorithm used to update the basis inverse. Due to the excellent sparcity and numerical stability characteristics, an LU factorization with either a Bartels-Golub or Forrest-Tomlin update has been adopted for modern linear programming systems. For pure network problems, the basis is always triangular and corresponds to a rooted spanning tree. The modern network codes which exploit this structure have been found to be from one to two orders of magnitude faster than the general linear programming systems. In this paper, we have combined these two powerful techniques into an algorithm for solving network models having additional side constraints.

Let A be an $\bar{m} \times \bar{n}$ matrix, let **c** and **u** be $\bar{n}$-component vectors, and let **b** be an $\bar{m}$-component vector. Without loss of generality, the **linear program** may be stated mathematically as follows:

minimize	$\mathbf{cx}$	(1)
subject to:	$\mathbf{Ax} = \mathbf{b}$	(2)
	$\mathbf{0} \leq \mathbf{x} \leq \mathbf{u}.$	(3)

The **network with side constraint model** is a special case of (1)–(3) in which A takes the form

$$A = \left[\begin{array}{c|c} M & \\ \hline S & P \end{array}\right] \begin{matrix} \} n \\ \} m \end{matrix}$$

where M is a node-arc incidence matrix.
If m = 0, then (1) – (3) is a pure network problem.

1.1 Applications

There are numerous applications of the network with side constraint model. Professor Glover and his colleagues have solved a large passenger-mix model for Frontier Airlines and a large land management model for the Bureau of Land Management (see [7, 8]). A world grain export model has been solved to help analyze the port capacity of U. S. ports during the next decade (see [2]). A cargo routing model is being used by the Air Force Logistics Command to assist in routing cargo planes for the distribution of serviceable spares (see [1]). Lt. Col. Dennis McLain, has developed a large model to assist in the development of a casualty evacuation plan in the event of a European conflict (see [14]). A National Forest Management Model has been developed to aid forest managers in long term planning for national forests (see [10]). In addition, work is currently underway which attempts to convert general linear programs into the network with side constraint model (see [4, 16]).

1.2 Objective of Investigation

Due to both storage and time considerations, the basis inverse is maintained as an LU factorization in modern LP software (see [3, 5, 15]). The objective of this investigation is to extend these ideas to the primal partitioning algorithm when applied to the network with side constraints model.

1.3 Notation

The i^{th}component of the vector **a** will be denoted by a_i. The $(i,j)^{th}$ element of the matrix A is denoted by A_{ij}. A(i) and A[i] denotes the i^{th} column and i^{th} row of the matrix A, respectively. **0** denotes a vector of zeroes, **1** denotes a vector of ones, and $\mathbf{e}^k$ denotes a vector with a 1 in the k^{th} position and zeroes elsewhere. Sigma is used to denote the scalar signum function defined by

$$\sigma(y) = \begin{cases} 1, \text{ if } y > 0 \\ 0, \text{ if } y = 0 \\ -1, \text{ if } y < 0. \end{cases}$$

The identity matrix is given by "I".

II. THE PRIMAL SIMPLEX ALGORITHM

We assume that A has full row rank and that there exist a feasible solution for (1)–(3). Given a basic feasible solution, we may partition A, $\mathbf{c}$, $\mathbf{x}$, and $\mathbf{u}$ into basic and nonbasic components, that is, $A = [B;N]$, $c = [\mathbf{c}^B;\mathbf{c}^N]$, $\mathbf{x} = [\mathbf{x}^B;\mathbf{x}^N]$, and $\mathbf{u} = [\mathbf{u}^B;\mathbf{u}^N]$. Using the above partitioning, the primal simplex algorithm may be stated as follows:

PRIMAL SIMPLEX ALGORITHM

0. *Initialization.* Let $[\mathbf{x}^B;\mathbf{x}^N]$ be a basic feasible solution.
1. *Pricing.* Let $\boldsymbol{\pi} = \mathbf{c}^B B^{-1}$. Define

   $\psi_1 = \{i : x_i^N = 0 \text{ and } \boldsymbol{\pi}\, N(i) > c_i^N\}$,

   $\psi_2 = \{i : x_i^N = u_i^N \text{ and } \boldsymbol{\pi}\, N(i) < c_i^N\}$.

   If $\psi_1 \cup \psi_2 = \emptyset$, terminate with $[\mathbf{x}^B;\mathbf{x}^N]$ optimal; otherwise, select $k \in \psi_1 \cup \psi_1$ and set $\delta \leftarrow 1$ if $k \in \psi_1$ and $\delta \leftarrow -1$, otherwise.
2. *Ratio Test.* Set $\mathbf{y} \leftarrow B^{-1}N(k)$. Set

   $$\Delta_1 \leftarrow \min_{\sigma(y_i)} = \delta\left\{\frac{x_i^B}{|y_i|}, \infty\right\}$$

   $$\Delta_2 \leftarrow -\min_{\sigma(y_i)} = \delta\left\{\frac{u_i^B - x_i^B}{|y_i|}, \infty\right\}$$

   Set $\Delta \leftarrow \min\{\Delta_1, \Delta_2, u_k^N\}$.
   If $\Delta \neq \infty$, then go to 3; otherwise, terminate with the conclusion that the problem is unbounded.
3. *Update Values.* Set $x_k^N \leftarrow x_k^N + \Delta\delta$ and $\mathbf{x}^B \leftarrow \mathbf{x}^B - \Delta\delta\mathbf{y}$. If $\Delta = u_k^N$, return to step 1.
4. *Update Basis Inverse.* Let

   $\psi_3 = \{j : x_i^B = 0 \text{ and } \sigma(y_i) = \delta\}$

   $\psi_4 = \{j : x_i^B = u_i^B \text{ and } -\sigma(y_i) = \delta\}$.

   Select any $\ell \in \psi_3 \cup \psi_4$. In the basis, replace $B(\ell)$ with $N(k)$, update the inverse of the new basis, and return to step 1.

III. THE PARTITIONED BASIS

The network with side constraint model may be stated as follows:

$$\text{minimize} \qquad \mathbf{c}^1\mathbf{x}^1 + \mathbf{c}^2\mathbf{x}^2 \tag{4}$$

$$\text{subject to:} \qquad M\mathbf{x}^1 = \mathbf{b}^1 \tag{5}$$

$$S\mathbf{x}^1 + P\mathbf{x}^2 = \mathbf{b}^2 \tag{6}$$

$$\mathbf{0} \le \mathbf{x}^1 \le \mathbf{u}^1 \tag{7}$$

$$\mathbf{0} \le \mathbf{x}^2 \le \mathbf{u}^2. \tag{8}$$

We may assume without loss of generality that,

(i) The graph associated with M has n nodes and is connected (i.e., there exists an undirected path between every pair of nodes)

(ii) [S¦P] has full row rank (i.e., rank [S¦P] = m).

(iii) Total supply equals total demand (i.e., $\mathbf{1}\mathbf{b}^1 = 0$).

Since the rank of system (5) is one less than the number of rows, we add what has been called the root arc to (5) to obtain

$$M\mathbf{x}^1 + \mathbf{e}^p a = \mathbf{b}^1$$

where $0 \le a \le 0$ and $1 \le p \le n$.

Then the constraint matrix for the network with side constraints model becomes

$$A = \left[\begin{array}{c|c|c} M & & \mathbf{e}^p \\ \hline S & P & \end{array}\right]$$

and A has full row rank

It is well-known that every basis for A may be placed in the form

$$B = \left[\begin{array}{c|c} T & C \\ \hline D & F \end{array}\right] \tag{9}$$

where T corresponds to a rooted spanning tree and

$$B^{-1} = \left[\begin{array}{c|c} T^{-1} + T^{-1}CQ^{-1}DT^{-1} & -T^{-1}CQ^{-1} \\ \hline -Q^{-1}DT^{-1} & Q^{-1} \end{array}\right] \tag{10}$$

where $Q = F - DT^{-1}C$. The objective of this paper is to give algorithms which maintain Q^{-1} as an LU factorization.

IV. THE INVERSE UPDATE

Recall that the partitioned basis takes the form

$$B = \left[\begin{array}{c|c} \underbrace{T}_{\text{key}} & \underbrace{C}_{\text{nonkey}} \\ \hline D & F \end{array}\right].$$

Let

$$L = \left[\begin{array}{c|c} T^{-1} & -T^{-1}C \\ \hline 0 & I \end{array}\right]$$

and let

$$\bar{B} = BL = \left[\begin{array}{c|c} I & 0 \\ \hline DT^{-1} & Q \end{array}\right].$$

The inverse update requires a technique for obtaining a new Q^{-1} after a basis exchange. Let $\bar{B}_i$, L_i, B_i, and Q_i denote the above matrices at iteration i. Then we want an expression for Q_{i+1}^{-1} in terms of Q_i^{-1}. The transformation takes the form

$$B_{i+1}^{-1} = EB_i^{-1} \tag{11}$$

where E is either an elementary column matrix or a permutation matrix. Let E be partitioned to be compatible with B. That is,

$$E = \left[\begin{array}{c|c} E_1 & E_2 \\ \hline E_3 & E_4 \end{array}\right] \begin{matrix} \}n \\ \}m \end{matrix}$$
$$\underbrace{}_{n} \quad \underbrace{}_{m}$$

By examining the (2,2) partition of $\bar{B}_{i+1}^{-1}$, we obtain

$$Q_{i+1}^{-1} = (E_4 - E_3T^{-1}C)Q_1^{-1} \tag{12}$$

In determining the updating formulae, we must examine two major cases with subcases.

Case 1. The leaving column is nonkey. For this case, E takes the form

$$\left[\begin{array}{c|c} 1 & E_2 \\ \hline 0 & E_4 \end{array}\right].$$

and (12) reduces to $Q_{i+1}^{-1} = E_4Q_i^{-1}$.

Case 2. The leaving column is key.

Let $\gamma = e^jT^{-1}C$. If $\gamma_k \neq 0$, then the k^{th} column of C can be interchanged with the j^{th} column of T and the new $T_=$ will be nonsingular.

Subcase 2a. $\gamma \neq 0$. Suppose $\gamma_k \neq 0$.

Then $E_4 - E_3T^{-1}C$ reduces to

$$R = \left[\begin{array}{c|c|c} I & & I \\ \hline & -e^jT^{-1}C & \\ \hline I & & I \end{array}\right] \begin{matrix} \\ \leftarrow \text{row } j \\ \text{, and} \end{matrix} \tag{13}$$

$Q_{i+1}^{-1} = RQ_i^{-1}$. Case 1 is applied to complete the update.

Subcase 2b. $\gamma = 0$. For this case no interchange is possible, the entering column becomes key, and $Q_{i+1}^{-1} = Q_i^{-1}$.

V. AN LU UPDATE

Let

$$U^i = \begin{bmatrix} I & \mu_1 & 0 \\ & \vdots & \\ & \mu_{m-1} & \\ & 1 & \\ 0 & & I \end{bmatrix}$$

and

$$L^i = \begin{bmatrix} I & & 0 \\ & \ell_i & \\ 0 & \ell_{i+1} & \\ & \vdots & I \\ & \ell_m & \end{bmatrix}$$

Matrices of the form given by U^i and L^i are called upper etas and lower etas, respectively. Suppose we have a factorization of Q^{-1} in the form

$$Q^{-1} = U^1U^2 \ldots U^mF^sF^{s-1} \ldots F^1, \tag{14}$$

where $F^1, \ldots, F^5$ are a combination of row and column etas. The right side of (14) is referred to as the eta file where only the non-identity rows and columns are stored. Suppose that the k^{th} column of Q is replaced by $\hat{Q}(k)$ to form the new m by m working basis $\hat{Q}$. This section presents algorithms which may be used to update (14) to produce $\hat{Q}^{-1}$ in the same form.

5.1 Nonkey Column Leaves The Basis

If $k = m$, then let $\beta = F^s \ldots F^1\hat{Q}(k)$, let

$$\tilde{L}^m = \begin{bmatrix} I & \\ & 1/\beta_m \end{bmatrix},$$

and let

$$\tilde{U}^m = \begin{bmatrix} I & -\beta_1 \\ & \vdots \\ & -\beta_{m-1} \\ & 1 \end{bmatrix}.$$

We will show that $\hat{Q}^{-1} = U^1 \ldots U^{m-1}\tilde{U}^m\tilde{L}^m F^s \ldots F^1$.

If $k < m$, then let $R^k = I$ and

$$Q^{-1} = U^1 \ldots U^k R^k U^{k+1} \ldots U^m F^s \ldots F^1. \tag{15}$$

We next define a new upper eta, $\tilde{U}^k$, and a new row eta, R^{k+1}, such that

$$R^k U^{k+1} = \tilde{U}^k R^{k+1}. \tag{16}$$

Substituting (16) into (15) yields

$$Q^{-1} = U^1 \ldots U^k \tilde{U}^k R^{k+1} U^{k+2} \ldots U^m F^s \ldots F^1. \tag{17}$$

We again define two new eta's, $\tilde{U}^{k+1}$ and R^{k+2}, such that

$$R^{k+1} U^{k+2} = \tilde{U}^{k+1} R^{k+2}. \tag{18}$$

Substituting (18) into (17) yields

$$Q^{-1} = U^1 \ldots U^k \tilde{U}^k \tilde{U}^{k+1} R^{k+2} U^{k+3} \ldots U^m F^s \ldots F^1.$$

Repeating this process eventually yields

$$Q^{-1} = U^1 \ldots U^k \tilde{U}^k \ldots \tilde{U}^{m-1} R^m F^s \ldots F^1. \tag{19}$$

Let $\boldsymbol{\gamma} = R^m F^s \ldots F^1 \hat{Q}(k)$, let

$$\tilde{L}^m = \begin{bmatrix} I & & & \\ & 1/\gamma_k & & \\ & -\gamma_{k+1}/\gamma_k & & \\ & \vdots & I & \\ & -\gamma_m/\gamma_k & & \end{bmatrix}$$

and let

$$\tilde{U}^m = \begin{bmatrix} I & -\gamma_1 & & \\ & \vdots & & \\ & -\gamma_{k-1} & & \\ & 1 & & \\ & & I & \end{bmatrix}$$

Then $\tilde{U}^m\tilde{L}^m\boldsymbol{\gamma} = \mathbf{e}^k$ and we will show that $\hat{Q}^{-1} = U^1 \ldots U^{k-1}\tilde{U}^k \ldots \tilde{U}^m\tilde{L}^m R^m F^s \ldots F^1$.

We now present the algorithm which updates the LU representation of Q^{-1} when the leaving column is nonkey. Assume that $\hat{Q}(k)$ is replacing $Q(k)$ in the working basis

ALG 1: LU UPDATE FOR NONKEY LEAVING COLUMN

1. Set $\boldsymbol{\beta} \leftarrow F^s \ldots F^1 \hat{Q}(k)$.
2. If $k \neq m$, set $\ell \leftarrow k$, $R^\ell \leftarrow I$, go to 4.
3. Set $\tilde{L}^m \leftarrow I$, where I is m by m.
   Set $\tilde{L}^m_{mm} \leftarrow 1/\beta_m$.
   Set $\tilde{U}^m \leftarrow I$, where I is m by m.
   Set $\tilde{U}^m_{im} \leftarrow -\beta_i$, for $1 \leq i < m$.
   Stop with $\hat{Q}^{-1} = U^1 \ldots U^{m-1}\tilde{U}^m\tilde{L}^m F^s \ldots F^1$.
4. Set $\alpha \leftarrow R^\ell[k]U^{\ell+1}(\ell+1)$.
   Set $R^{\ell+1} \leftarrow R^\ell$.
   Set $R^{\ell+1}_{k\ell+1} \leftarrow \alpha$.
   Set $\tilde{U}^\ell \leftarrow U^{\ell+1}$.
   Set $\tilde{U}^\ell_{k\ell+1} \leftarrow 0$.
   $(R^\ell U^{\ell+1} = \tilde{U}^\ell R^{\ell+1})$
   Set $\ell \leftarrow \ell+1$.
5. If $\ell < m$, go to 4.
   $(U^{k+1} \ldots U^m = \tilde{U}^k \ldots \tilde{U}^{m-1} R^m.)$
   Set $\boldsymbol{\beta} \leftarrow R^m\boldsymbol{\beta}$.
6. Set $\tilde{L}^m \leftarrow I$, where I is m by m.
   Set $\tilde{L}^m_{kk} \leftarrow 1/\beta_k$.
   Set $\tilde{L}^m_{jk} \leftarrow -\beta_j/\beta_k$, for $k < j \leq m$.
   Set $\tilde{U}^m \leftarrow I$, where I is m by m.
   Set $\tilde{U}^m_{ik} \leftarrow -\beta_i$, for $1 \leq i \leq k$.
   Set $\tilde{U}^m_{kk} \leftarrow 1$.
   Stop with $\hat{Q}^{-1} = U^1 \ldots U^{k-1}\tilde{U}^k\tilde{U}^{k+1} \ldots \tilde{U}^m\tilde{L}^m R^m F^s \ldots F^1$.

We now present the justification for step 3 of ALG 1. For $k = m$, we claim that $\hat{Q}^{-1} = U^1 \ldots U^{m-1}\tilde{U}^m\tilde{L}^m F^s \ldots F^1$. Note that $\hat{Q}^{-1}\hat{Q}(m) = U^1 \ldots U^{m-1}\tilde{U}^m\tilde{L}^m\boldsymbol{\beta}$. But by construction $\tilde{U}^m\tilde{L}^m\boldsymbol{\beta} = \mathbf{e}^m$. Consider

Proposition 1.

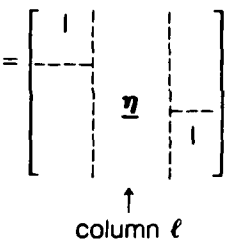
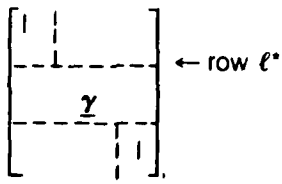
Let $\boldsymbol{\beta}$ be any m-vector and E' be any column eta. If $\beta_i = 0$, then $E'\boldsymbol{\beta} = \boldsymbol{\beta}$.

By Proposition 1, $U^1 \ldots U^{m-1}\mathbf{e}^m = \mathbf{e}^m$. Therefore, $\hat{Q}^{-1}\hat{Q}(m) = \mathbf{e}^m$. For $1 \leq \ell < m$, let $\boldsymbol{\gamma} = F^s \ldots F^1 Q(\ell)$. By construction $\gamma_j = 0$ for $\ell < j \leq m$ and $\gamma_\ell = 1$. By Proposition 1, $U^{\ell+1} \ldots U^{m-1}\tilde{U}^m\tilde{L}^m\boldsymbol{\gamma} = \boldsymbol{\gamma}$. By the construction of $U^1 \ldots U^\ell$, we have $U^1 \ldots U^\ell\boldsymbol{\gamma} = \mathbf{e}^\ell$. Therefore, if the leaving column is Q(m), then step 3 of ALG 1 produces $\hat{Q}^{-1}$.

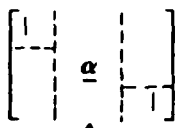
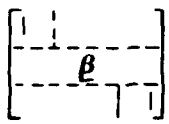
We now present a theoretical justification for step 4 of ALG 1.

Proposition 2.

Let

$U^{p+1} =$  and $R^p =$  ← row ℓ^*

where $\ell \neq \ell^*$.
If

$\tilde{U}^p =$  and $R^{p+1} =$  ← row ℓ^*

where

$$\alpha_i = \begin{cases} 0, \text{ if } i = \ell^* \\ n_i, \text{ otherwise, and} \end{cases}$$

$$\beta_i = \begin{cases} \mathbf{n}\boldsymbol{\gamma}, \text{ if } i = \ell \\ \gamma_i, \text{ otherwise,} \end{cases}$$

then $R^pU^{p+1} = \tilde{U}^pR^{p+1}$.

Proposition 2 is a theoretical justification for step 4 of ALG 1. The proposition to follow shows the precise structure of $R^mF^s \ldots F^1Q$. Consider

Proposition 3.

Let $U^* = F^s \ldots F^1Q$. If $\tilde{U}^* = R^mU^*$, then

$$\tilde{U}^*[i] = \begin{cases} U^*[i], \ i \neq k \\ \mathbf{e}^k, \text{ otherwise.} \end{cases}$$

We now present the results to prove that $\hat{Q}^{-1} = U^1 \ldots U^{k-1}\tilde{U}^k \ldots \tilde{U}^m\tilde{L}^mR^mF^s \ldots F^1$.

Proposition 4.

$$U^1 \ldots U^{k-1}\tilde{U}^k \ldots \tilde{U}^m\tilde{L}^mR^mF^s \ldots F^1Q(k) = \mathbf{e}^k.$$

Proposition 5.

$$U^1 \ldots U^{k-1}\hat{U}^k \ldots \tilde{U}^m\tilde{L}^mR^mF^s \ldots F^1\hat{Q}(i) = \mathbf{e}^i \text{ for } i \neq k.$$

By Propositions 4 and 5, we have

Corollary 6.

$$\hat{Q}^{-1} = U^1 \ldots U^{k-1}\hat{U}^k \ldots \tilde{U}^m\tilde{L}^m R^m F^s \ldots F^1.$$

Hence, ALG 1 produces the updated working basis inverse.

5.2 Key Column Leaves The Basis

In this section, we present an algorithm for updating the working basis inverse to accomplish a switch between a key column and a nonkey column. That is, $\hat{Q} = RQ^{-1}$ where R is given by (13) and

$$Q^{-1} = U^1 \ldots U^m F^s \ldots F^1. \tag{20}$$

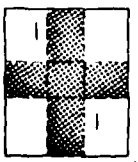
We wish to obtain $\hat{Q}^{-1}$ in the same form as (20)

To accomplish this update, we begin with $Q^{-1} = RU^1 \ldots U^m F^s \ldots F^1$. We apply Proposition 2 to RU^1 creating the factorization $\hat{Q}^{-1} = \tilde{U}^1 R^2 U^2 \ldots U^m F^s \ldots F^1$. We continue with the application of Proposition 2 until we obtain $\hat{Q}^{-1} = \tilde{U}^1 \ldots \tilde{U}^{k-1} R^k U^k \ldots U^m F^s \ldots F^1$. Proposition 2 does not apply to $R^k U^k$. However, a simple update would be to let $\hat{U}^k = \ldots = \hat{U}^m = I$ and use the below factorization:

$$\hat{Q}^{-1} = \underbrace{\tilde{U}^1 \quad \hat{U}^m}_{\text{LEFT FILE}} R^k \underbrace{U^k \quad U^m F^s \quad F^1}_{\text{RIGHT FILE}}.$$

This update simply involves application of Proposition 2 until it does not apply ($\ell = \ell^*$) and then shifting the remainder of the left file to the right file. We call this update the *TYPE 1 UPDATE*.

We will now give an update in which $R^k U^k \ldots U^m$ is modified as opposed to moving them to the right file. Let

$E^k = R^k U^k =$  ← row k

Then we define matrices $\tilde{U}^{k+1}$ and E^{k+1} such that $E^k U^{k+1} = \tilde{U}^{k+1} E^{k+1}$. Following this procedure, $R^k U^k \ldots U^m$ can be replaced by $\tilde{U}^{k+1} \ldots \tilde{U}^m E^{m+1}$ so that

$$\hat{Q}^{-1} = \tilde{U}^1 \ldots \tilde{U}^{k-1}\tilde{U}^{k+1} \ldots \tilde{U}^m E^{m+1} F^s \ldots F^1.$$

Further, we define a row eta $\tilde{R}$ and a column eta $\tilde{F}$ such that $E^{m+1} = \tilde{R}\tilde{F}$. Therefore,

$$\hat{Q}^{-1} = \underbrace{\tilde{U}^1 \quad \tilde{U}^{k-1}\tilde{U}^{k+1} \quad \hat{U}^m}_{\text{LEFT FILE}} \underbrace{\tilde{R}\tilde{F}F^s \quad F^1}_{\text{RIGHT FILE}}.$$

We call this update the *TYPE 2 UPDATE*.

We now present a set of propositions which justify the *TYPE 2 UPDATE*

Proposition 7.

Let

$$U^{p+1} = \begin{array}{|c|c|c|} \hline I & \begin{matrix}\eta_1 \\ \vdots \\ \eta_{\ell-1}\end{matrix} & 0 \\ \hline & \eta_\ell & \\ \hline 0 & \begin{matrix}\eta_{\ell+1} \\ \vdots \\ \eta_n\end{matrix} & I \\ \hline \end{array} \quad \text{and } E^p = \begin{array}{|c|c|c|} \hline I & \begin{matrix}\mu_1 \\ \vdots \\ \mu_{\ell^*-1}\end{matrix} & 0 \\ \hline \gamma_1 \cdots \gamma_{\ell^*-1} & \gamma_{\ell^*} & \gamma_{\ell^*+1} \cdots \gamma_n \\ \hline 0 & \begin{matrix}\mu_{\ell^*+1} \\ \vdots \\ \mu_n\end{matrix} & I \\ \hline \end{array}$$

where $\ell \neq \ell^*$ and $\mu_\ell = 0$.

If

$$\tilde{U}^{p+1} = \begin{array}{|c|c|c|} \hline I & \begin{matrix}\alpha_1 \\ \vdots \\ \alpha_{\ell-1}\end{matrix} & 0 \\ \hline & \alpha_\ell & \\ \hline 0 & \begin{matrix}\alpha_{\ell+1} \\ \vdots \\ \alpha_n\end{matrix} & I \\ \hline \end{array} \quad \text{and } E^{p+1} = \begin{array}{|c|c|c|} \hline I & \begin{matrix}\mu_1 \\ \vdots \\ \mu_{\ell^*-1}\end{matrix} & \\ \hline \lambda_1 \cdots \lambda_{\ell^*-1} & \lambda_{\ell^*} & \lambda_{\ell^*+1} \cdots \lambda_n \\ \hline 0 & \begin{matrix}\mu_{\ell^*+1} \\ \vdots \\ \mu_n\end{matrix} & I \\ \hline \end{array}$$

where

$$\lambda_i = \begin{cases} \gamma\eta, \text{ if } i = \ell, \\ \gamma_i, \text{ otherwise,} \end{cases}$$

$$\alpha_i = \begin{cases} 0, \text{ if } i = \ell^*, \\ \eta_i + \mu_i\eta_{\ell^*}, \text{ otherwise,} \end{cases}$$

then $E^pU^{p+1} = \tilde{U}^{p+1}E^{p+1}$.

The following proposition is used to replace the cross matrix E^{m+1} with a row eta $\hat{R}$ and a column eta $\hat{F}$.

Proposition 8.
Let

$$E = \left[\begin{array}{c|c|c} I & \begin{matrix}\mu_1\\ \vdots\\ \mu_{\ell-1}\end{matrix} & 0 \\ \hline \gamma_1 \cdots \gamma_{\ell-1} & \gamma_\ell & \gamma_{\ell+1} \cdots \gamma_n \\ \hline 0 & \begin{matrix}\mu_{\ell+1}\\ \vdots\\ \mu_n\end{matrix} & I \end{array}\right]$$

If

$$\hat{R} = \left[\begin{array}{c|c|c} I & & 0 \\ \hline \gamma_1 \cdots \gamma_{\ell-1} & X & \gamma_{\ell+1} \cdots \gamma_n \\ \hline 0 & & I \end{array}\right] \quad \text{and} \quad \hat{F} = \left[\begin{array}{c|c|c} I & \begin{matrix}\mu_1\\ \vdots\\ \mu_{\ell-1}\end{matrix} & 0 \\ \hline & Y & \\ \hline 0 & \begin{matrix}\mu_{\ell+1}\\ \vdots\\ \mu_n\end{matrix} & I \end{array}\right]$$

where X and Y are such that

$$XY = \gamma_\ell - \sum_{\substack{i=1\\ i\neq\ell}}^{n} \gamma_i \mu_i$$

then $E = \hat{R}\hat{F}$
We now present the update algorithm for the case in which the ℓ^{th} column of T is being switched with the k^{th} column of C. Let $\boldsymbol{\gamma} = \mathbf{e}'T^{-1}C$

ALG 2 LU UPDATE FOR A KEY LEAVING COLUMN

1 Set $R^1 \leftarrow I$
  Set $R^1[k] \leftarrow \boldsymbol{\gamma}$
  Set $i \leftarrow 1$
2 If $i = k$, go to 4
  Set $\alpha \leftarrow R^i[k]U^i(i)$
  Set $R^{i+1} \leftarrow R^i$
  Set $R^{i+1}_k \leftarrow \alpha$
  Set $U^{i+1} \leftarrow U^i$
  Set $U^{i+1}_k \leftarrow 0$
3 Set $i \leftarrow i + 1$ and go to 2
4 Set $U^n \leftarrow I$
  Set $E^n \leftarrow R^n U^n$

5. Apply Proposition 7 to $E^i U^{i+1}$ to form $\tilde{U}^{i+1}\tilde{E}^{i+1}$.
   Set $i \leftarrow i + 1$.
6. If $i < m$, go to 5.
7. Apply Proposition 8 to E^m to obtain $\tilde{R}\tilde{F}$ where $X = 1$.
   At the completion of step 7 we have $\hat{Q}^{-1} = \tilde{U}^1 \ldots \tilde{U}^m \tilde{R}\tilde{F}F^s \ldots F^1$.

VI. COMPUTATIONAL EXPERIMENTATION

Three test problems were selected for the experiment. Sc205 is a staircase linear program which was generated by Ho and Loute [12] and transformed into a network with side constraints. Gifford-Pinchot is a model of the Gifford-Pinchot National Forest [10] which has also been transformed into a network with side constraints. RAN is a randomly generated problem.

These problems were first solved and the pivot agenda was saved. That is, entering and leaving columns for each pivot were saved on a file. This file was then used by each code so that all three basis updates follow the same path to the optimum. The number of nonzeroes required to represent Q^{-1} at various points in the solution process is illustrated in Figures 1 and 2. For both problems, the LU Type 2 update dominated both the LU Type 1 update and the product-form code in terms of nonzeroes in the inverse. The average core storage required for Q^{-1} using the product-form update is approximately double that required for the best LU update.

Given the above results, we developed three specialized network with side constraints codes and computationally compared them with three general in-core LP systems and a special system for multicommodity network flow problems. All codes are written in FORTRAN and have not been tailored to either our equipment or our FORTRAN compiler. None of the codes were tuned for our problem set. A brief description of each code follows.

NETSIDE1, NETSIDE2 AND NETSIDE3 are our specialized network with side constraints systems. The first maintains Q^{-1} in product form, while the second and third maintain Q^{-1} in LU form using a Type 1 and Type 2 update, respectively. All use the Hellerman and Rarick [11] reinversion routine. The working basis is reinverted every 60 iterations. The pricing routine uses a candidate list of size 6 with block size of 200.

MINOS [15] stands for "a Modular In-Core Nonlinear Optimization System" and is designed to solve problems of the following form.

$$\begin{aligned} \text{minimize} \quad & \mathbf{f(x)} + \mathbf{cx} \\ \text{subject to:} \quad & \mathbf{Ax} = \mathbf{b} \\ & \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

where $\mathbf{f(x)}$ is continuously differentiable in the feasible region. For this

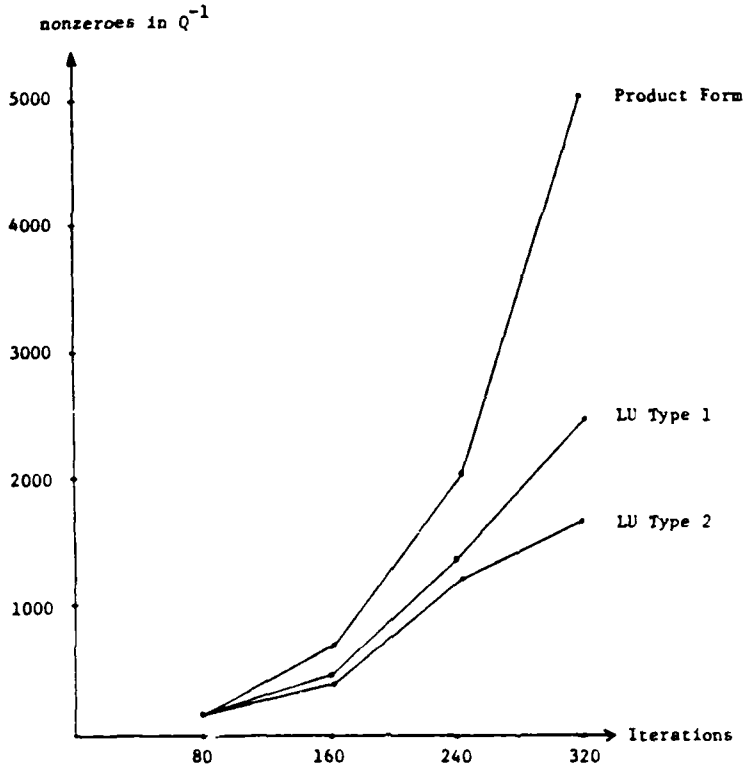


Figure 1. Nonzero Buildup In The Working Basis Inverse On SC205 [22].
(317 columns, 119 nodes, 87 side constraints)

study f(x) = 0 at all **x** and therefore none of the nonlinear subroutines were used for problem solution.

For linear programs, MINOS uses the revised simplex algorithm with all data and instructions residing in core storage. The basis inverse is maintained as an LU factorization using a Bartels-Golub update. The reinversion routine uses the Hellerman-Rarick [11] pivot agenda algorithm.

XMP is a library of FORTRAN subroutines which can be used to solve linear programs. The basis inverse is maintained in LU factored form. The pricing routine uses a candidate list of size 6 with two hundred columns being scanned each time the list is refreshed. The basis is reinverted every 50 iterations.

LISS stands for "Linear In-Core Simplex System" and is an in-core LP solver with the basis inverse maintained in product form. The reinversion routine is a modification of the work of Hellerman and Rarick [11]. The basis inverse is refactored every 50 iterations. A partial pricing scheme is used with 20 blocks

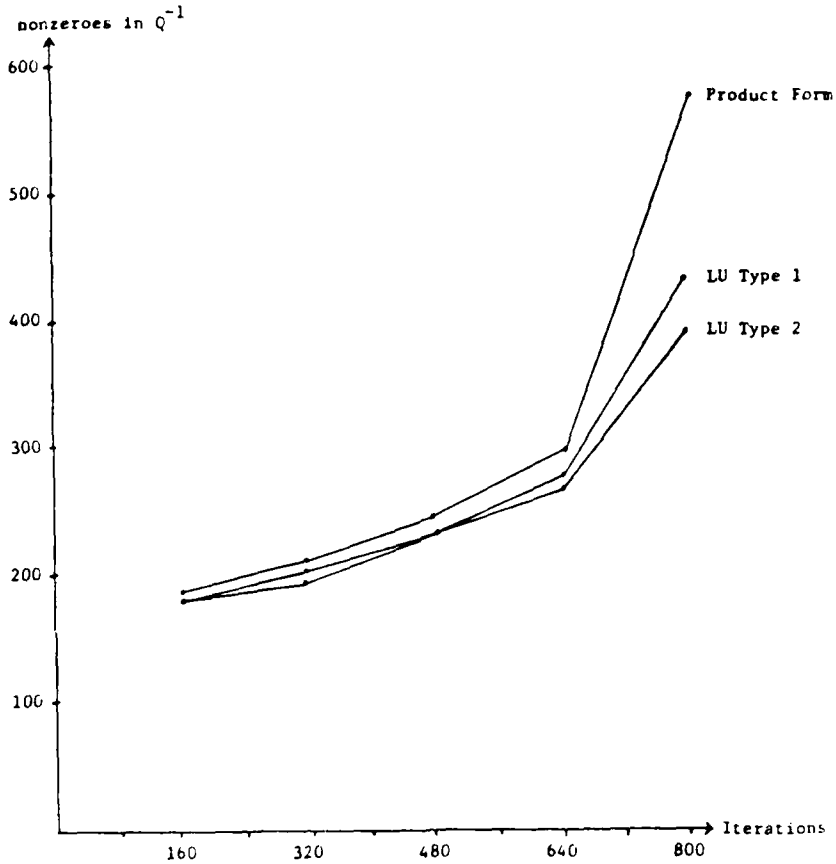


Figure 2. Nonzero Buildup In The Working Basis Inverse On Gifford Pinchot [20].
(1160 columns, 533 nodes, 84 side constraints)

MCNF stands for "Multicommodity Network Flow". MCNF uses the primal partitioning algorithm also. The basis inverse is maintained as a set of rooted spanning trees (one for each commodity) and a working basis inverse in product form. This working basis inverse has dimension equal to the number of binding GUB constraints. A partial pricing scheme is used. Our computational experience is given in Table 1.

The row entitled GUB Constraints, gives the number of LP rows which correspond to "GUB Constraints". The row, entitled "Binding GUB Constraints", gives the number of GUB constraints met as equalities at optimality using MCNF. All runs were made on the CDC 6600 at Southern Methodist University using the FTN compiler with the optimization feature enabled.

Table 1 **Comparison of Codes for Solving Multicommodity Network Flow Problems**
(All Times Exclude Input and Output)

PROB DESC																					
Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Totals
LP Rows	500	400	401	501	499	415	576	561	547	496	559	477	513	490	532	552	544	571	544	577	
LP Cols	995	720	841	896	851	732	850	972	901	900	934	842	843	840	840	848	900	841	872	841	
% Network Rows	100	100	99 4	99 4	80	96	56	86	25	32	89	63	47	56	45	62	63	66	54	73	
GUB Const	0	0	1	1	99	15	256	81	412	336	59	177	273	215	292	212	204	196	254	157	
Binding GUB Const	0	0	0	0	2	3	5	3	6	6	7	9	8	11	13	17	21	23	26	31	
Number Nonzeroes	1910	1440	1701	1794	2553	1746	2550	2916	2703	2700	2104	2526	2529	2520	2394	2414	2700	2523	2616	2523	
Number Commodities	5	20	20	5	10	20	4	12	3	4	5	6	4	5	3	4	4	5	3	6	
MINOS [31]																					
Scaled Time	984	810	839	862	563	688	386	633	484	517	451	361	329	231	356	281	198	130	172	154	
Time (sec)	99 40	43 07	44 29	82 61	56 70	41 86	61 76	83 66	29 09	31 70	79 56	44 63	42 55	44 19	45 28	55 86	62 92	63 35	50 72	67 39	1131
Pivots	1207	692	684	1068	739	655	750	978	376	427	986	619	564	592	594	712	781	793	692	831	
XMP [28]																					
Scaled Time	632	582	559	620	426	528	356	498	452	507	435	376	269	272	381	329	207	189	222	275	
Time (sec)	63 81	30 98	29 49	59 43	42 94	32 12	57 06	65 78	27 17	31 10	76 70	46 48	34 90	52 11	48 43	67 53	65 85	92 11	65 73	120 38	1110
Pivots	1198	751	720	1109	806	737	945	1135	499	619	1243	906	661	945	877	1099	1117	1375	1085	1687	
LISS [2]																					
Scaled Time	398	326	337	364	433	454	565	622	1125	1503	400	1223	675	627	709	588	360	380	334	493	
Time (sec)	40 20	17 34	17 79	34 88	43 63	27 63	90 55	82 24	67 62	92 13	70 53	151 15	87 36	120 04	90 19	116 87	114 78	185 46	98 85	215 86	1765
Pivots	1267	766	769	1133	1319	1084	2087	2091	1799	2220	1767	2416	2048	2309	2137	2233	2400	2660	2120	2762	
MCNF [26]																					
Scaled Time	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	
Time (sec)	10 10	5 32	5 28	9 58	10 08	6 08	16 02	13 22	6 01	6 13	17 63	12 36	12 95	19 15	12 72	19 89	31 84	48 84	29 54	43 81	337
Pivots	722	493	451	717	466	443	595	610	212	246	745	530	515	677	461	662	989	1272	885	1191	
NETSIDE1																					
Scaled Time	230	240	235	245	177	209	185	201	285	243	159	167	146	145	203	143	108	74	126	111	
Time (sec)	23 18	12 79	12 41	23 43	17 87	12 68	29 64	26 51	17 11	14 87	28 06	20 63	18 96	27 85	25 87	28 50	34 25	36 31	37 33	48 65	720
Pivots	885	602	570	887	628	577	662	852	298	309	917	613	465	716	547	703	853	885	816	1180	
NETSIDE2																					
Scaled Time	229	239	230	242	182	209	191	200	296	253	161	167	152	148	209	148	111	76	134	110	
Time (sec)	23 17	12 72	12 16	23 23	18 34	12 72	30 61	26 42	17 77	15 48	28 44	20 61	19 71	28 40	26 59	29 46	35 49	37 23	39 49	48 40	731
Pivots	885	602	570	887	628	577	662	852	298	309	917	613	465	716	547	703	853	885	823	1177	
NETSIDE3																					
Scaled Time	232	247	234	247	179	212	195	202	292	255	163	171	155	158	212	150	115	79	136	117	
Time (sec)	23 41	13 16	12 38	23 69	18 00	12 86	31 19	26 73	17 56	15 65	28 78	21 17	20 03	30 25	17 01	29 74	36 74	38 37	40 22	51 38	816
Pivots	885	602	570	887	628	577	682	852	298	309	917	613	465	716	547	703	853	885	823	1180	

Based on these results, we conclude that for lightly constrained multicommodity network flow problems

(i) XMP and MINOS run at approximately the same speed,

(ii) NETSIDE1, NETSIDE2 and NETSIDE3 run at approximately the same speed, and

(iii) the three NETSIDE codes are approximately twice as fast as XMP and MINOS.

References

1. Ali, A., R. Helgason, and J. Kennington, "An Air Force Logistics Decision Support System Using Multicommodity Network Models", Technical Report 82-OR-1, Department of Operations Research, Southern Methodist University, Dallas, Texas 75275, (1982).
2. Barnett, D., J. Binkley, and B. McCarl, "The Effects of U. S. Port Capacity Constraints on National and World Grain Shipments", Technical Paper, Purdue Agricultural Experiment Station, Purdue University, West Lafayette, Indiana, (1982).
3. Bartels, R., and G. Golub, "The Simplex Method of Linear Programming Using LU Decomposition", **Communications of ACM,** 12, 266–268, (1969).
4. Bixby, R. E., "Recent Algorithms for Two Versions of Graph Realization and Remarks on Applications to Linear Programming", Technical Report.
5. Forrest, J. J. H., and J. A. Tomlin, "Updated Triangular Factors of the Basis to Maintain Sparcity in the Product Form Simplex Method", **Mathematical Programming,** 2, 3, 263–278, (1972).
6. Glover, F., and D. Klingman, "The Simplex Son Algorithm for LP/Embedded Network Problems", Technical Report CCS 317, Center for Cybernetic Studies, The University of Texas, Austin, Texas, (1977).
7. Glover, F., R. Glover, J. Lorenzo, and C. McMillan, "The Passenger-Mix Problem in the Scheduled Airlines", **Interfaces,** 12, 3, 73–80, (1982).
8. Glover, F., R. Glover, and F. Martinson, "The U. S. Bureau of Land Management's New Netform Vegetation Allocation System", Technical Report, Division of Information Science Research, University of Colorado, Boulder, Colorado, (1982)
9. Graves, G. W., and R. D. McBride, "The Factorization Approach to Large-Scale Linear Programming", **Mathematical Programming,** 10, 1, 91–110, (1976).

10. Helgason, R., J. Kennington, and P. Wong, "An Application of Network Programming for National Forest Planning", Technical Report OR 81006, Department of Operations Research, Southern Methodist University, Dallas, Texas, (1981).
11. Hellerman, E., and D. Rarick, "Reinversion With the Preassigned Pivot Procedure", **Mathematical Programming,** 1, 195–216, (1971).
12. Ho, J. K., and E. Loute, "A Set of Staircase Linear Programming Test Problems", **Mathematical Programming,** 20, 2, 245–250, (1981)
13. Kennington, J. L., and R. V. Helgason, **Algorithms for Network Programming,** John Wiley and Sons, New York, New York, (1980).
14. McLain, D. R., "A Multicommodity Approach to a Very Large Aeromedical Transportation Problem", (working paper) Operations Research Division, Military Airlift Command, Scott Air Force Base, Illinois, (1983).
15. Murtagh, B., and M. Saunders, "MINOS User's Guide", Technical Report 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, (1977).
16. Wagner, D. K., "An Almost Linear-Time Graph Realizat on Algorithm", unpublished dissertation, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois, (1983).

JEFFERY L. KENNINGTON

Jeffery L. Kennington is a professor and chairman of the Operations Research Department of Southern Methodist University. He received his Ph.D. from the Department of Industrial and Systems Engineering at Georgia Institute of Technology. He is a co-author of the John Wiley book entitled ***Algorithms for Network Programming***. His other publications have appeared in ***Mathematical Programming, Operations Research, Management Science, Naval Research Logistics Quarterly, Networks,*** and ***Institute of Industrial Engineering Transactions.***

RICHARD S. BARR

Richard S. Barr is Associate Professor of Operations Research at the School of Engineering and Applied Science, Southern Methodist University, Dallas, TX. He received his B.S. in Electrical Engineering, M.B.A. and Ph.D. in Operations Research, all from the University of Texas in Austin. His current research interest include ultra-large scale mathematical programming, with an emphasis on network optimization; microcomputer applications of operations research; microeconomic simulation models; and algorithms for new computer architectures. He is a contributing editor for ***Interfaces***, and has published in ***Operations Research, Mathematical Programming,*** and ***European Journal of Operational Research.***

KEYVAN FARHANGIAN

Keyvan Farhangian is a systems designer with Consilium Associates, Inc. of Palo Alto, CA. He received his B.A. in Business Administration from the University of Tehran, Iran, and his M.S. and Ph.D. in Operations Research from Southern Methodist University.

CHAPTER 3

The Frequency Assignment Problem: A Solution via Nonlinear Programming*

J. David Allen
Switching Systems Division, Rockwell International, P.O. Box 10462, Dallas, Texas 75207

Richard V. Helgason and Jeffery L. Kennington
Operations Research Department, Southern Methodist University, Dallas, Texas 75275

This paper gives a mathematical programming model for the problem of assigning frequencies to nodes in a communications network. The objective is to select a frequency assignment which minimizes both cochannel and adjacent-channel interference. In addition, a design engineer has the option to designate key links in which the avoidance of jamming due to self interference is given a higher priority. The model has a nonconvex quadratic objective function, generalized upper-bounding constraints, and binary decision variables. We developed a special heuristic algorithm and software for this model and tested it on five test problems which were modifications of a real-world problem. Even though most of the test problems had over 600 binary variables, we were able to obtain a near optimum in less than 12 seconds of CPU time on a CDC Cyber-875.

1. INTRODUCTION

One of the most critical design problems in a radio communication network is the assignment of transmit frequencies to stations (nodes) so that designated key communication links will not be jammed due to self interference. In this investigation, we describe a novel new optimization model and a solution technique which can be used to assist design engineers in this process.

1.1. Problem Description

A radio communications network consists of radio stations, each equipped with one or more transmitters and receivers. When a given station has the ability to receive information intelligibly from a transmitting station, a link is said to exist from the transmitting station to the receiving station. The interconnection of these stations and links may be viewed graphically as a set of nodes, representing the radio stations, joined together by directed arcs, representing the links.

We assume in our model that one transmitter and several receivers are located at each radio station (node). The transmitter is tuned to a specified center frequency, and the receivers are tuned to the transmit frequencies of the neighboring stations to which the station is to be linked. A channel is associated with each

*Comments and criticisms from interested readers are cordially invited.

Naval Research Logistics, Vol. 34, pp. 133–139 (1987)
 CCC 0028-1441/87/010133-07$04.00

center frequency in a way similar to the way channels and frequencies are associated in a television set. When a TV is tuned to channel 4, for example, it is really being tuned to receive video signals being broadcast at 67.25 MHz.

For our model, a given center frequency will be associated with each channel number. Using this definition, the frequency assignment problem may be defined as follows: Given N transmitting stations (nodes), assign 1 of F transmit channels to each node in such a way as to minimize the number of designated key links jammed due to cochannel and adjacent-channel interference. We say that a link is jammed if either of the following conditions occurs: (i) a node receives two signals on the same channel that are less than α dB apart in signal strength, or (ii) a node receives a signal on a given channel while a neighboring node transmits on an adjacent channel. If the neighbor's signal strength exceeds the signal strength of the current node by more than β dB, then the incoming signal will be garbled. The constants α and β are functions of the hardware used in the network. Some of the determining factors are the receiver selectivity, the type of signal modulation, and the purity of the signal.

We now introduce the notation used to describe the mathematical model. Let $f \in \{1, \ldots, F\}$ denote a channel and $n \in \{1, \ldots, N\}$ denote a node. $\mathbf{e}_i$ will denote a vector whose entries are 0 except for the ith, which is 1. Let $x_{fn} = 1$ if channel f is assigned to node n and 0 otherwise, $\mathbf{x}_f'$ = the row vector $[x_{f1}, \ldots, x_{fN}]$, and $g(\mathbf{x}_1, \ldots, \mathbf{x}_F)$ = a weighted number of jammed links with assignment $(\mathbf{x}_1, \ldots, \mathbf{x}_F)$. Using the above notation, the mathematical model of the frequency assignment problem is

$$\min \quad g(\mathbf{x}_1, \ldots, \mathbf{x}_F) \tag{1}$$

$$\text{s.t.} \quad \sum_f x_{fn} = 1, \quad \text{for all } n \tag{2}$$

$$x_{fn} \in \{0,1\}, \quad \text{for all } f,n. \tag{3}$$

For this application, $g(\cdot)$ is a nonconvex quadratic function and therefore (1)–(3) are members of the class of binary nonconvex cost nonlinear programs.

1.2. Related Literature

A heuristic procedure for solving a similar problem using a graph coloring algorithm has been evaluated by Zoellner and Beall [7]. Closely related models have been investigated by Morito, Salkin, and Williams [5] and by Mathur, Salkin, Nishimura, and Morito [4]. Their models are general linear integer programs with a single constraint. Using a special branch-and-bound algorithm, they successfully solved their model with up to fifty channels.

1.3. Accomplishments of the Investigation

We developed a novel new mathematical model of the frequency assignment problem which takes the form of a binary nonconvex quadratic cost nonlinear program. The model incorporates weighting constants that allow a design engineer to tune the model to a particular application. We present an elegant specialization of the convex simplex algorithm to obtain a local optimum for this model. In addition, specialized software has been developed for this model and tested on

five versions of a real-world problem. The software works quite well, requiring less than a minute of computer time for all five test problems.

2. THE OBJECTIVE FUNCTION

In this section, we define the weighted interference function, $g(\mathbf{x}_1, \ldots, \mathbf{x}_F)$. This function is generated from a set of signal strength matrices, $(A_1, \ldots, A_F)$, two weighting matrices, and a set of critical values α, β, and $\delta_1, \ldots, \delta_N$. Let a_{ij}^f denote the received signal strength in dBu/m of a signal which originates at node i and is received by node j, and let A_f denote the matrix whose elements are a_{ij}^f. Let the weighting matrices P and W be determined as follows:

$$p_{ij} = \begin{cases} \bar{p}_1, & \text{if } (i,j) \text{ is a designated key link} \\ \bar{p}_2, & \text{otherwise} \end{cases}$$

and

$$w_{ij} = \begin{cases} \bar{w}_1, & \text{if } (i,j) \text{ is a designated key link} \\ \bar{w}_2, & \text{otherwise.} \end{cases}$$

The constants $\bar{p}_1$, $\bar{p}_2$, $\bar{w}_1$, and $\bar{w}_2$ are tuning parameters which are used to provide weights in the interference function for the key links.

Gamma is used to denote the scalar function, defined by

$$\gamma(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Using $\gamma(\cdot)$, we define the three matrices

$$q_{ij}^f = \begin{cases} \displaystyle\sum_{\substack{k \neq i,j \\ \sigma_{ik}^f > \delta_k}} \gamma(\alpha - |a_{jk}^f - a_{ik}^f|) w_{ik} + p_{ij}, & i \neq j \\ 0, & \text{otherwise.} \end{cases}$$

$$r_{ij}^f = \begin{cases} \displaystyle\sum_{\substack{k \neq i,j \\ \sigma_{ik}^f > \delta_k}} \gamma(a_{jk}^{f-1} - a_{ik}^f - \beta) w_{ik}, & i \neq j \\ 0, & \text{otherwise.} \end{cases}$$

and

$$s_{ij}^f = \begin{cases} \displaystyle\sum_{\substack{k \neq i,j \\ \sigma_{ik}^f > \delta_k}} \gamma(a_{jk}^{f-1} - a_{ik}^f - \beta) w_{ik}, & i \neq j, \\ 0, & \text{otherwise.} \end{cases}$$

Using these matrices, the interference function is given by

$$g(\mathbf{x}_1, \ldots, \mathbf{x}_F) = \underbrace{\sum_{f=1}^{f=F} \mathbf{x}_f' Q_f \mathbf{x}_f}_{\substack{\text{cochannel} \\ \text{interference}}} + \underbrace{\sum_{f=1}^{f=F-1} \mathbf{x}_f' R_f \mathbf{x}_{f+1}}_{\substack{\text{adjacent channel} \\ \text{interference from} \\ \text{channel above}}} + \underbrace{\sum_{f=2}^{f=F} \mathbf{x}_f' S_f \mathbf{x}_{f-1}}_{\substack{\text{adjacent channel} \\ \text{interference from} \\ \text{channel below}}}.$$

In addition it is often desirable to use all of the channels. Therefore, we appended the function

$$\overline{w}_3 \sum_f \sum_{i=1}^{i=N-1} x_{fi} \sum_{j=i+1}^{j=N} x_{fj}$$

to $g(\cdot)$ so that in the absence of self interference, the channels would be equally distributed among the nodes. The scalar $\overline{w}_3$ is also a tuning parameter.

Using the above formulae, we now give an example which presents the matrices required to define $g(\cdot)$. Let $\alpha = 2$, $\beta = 3$, $\overline{w}_3 = 0$, $\delta_n = 0$ for all n, and $p_{ij} = w_{ij} = 1$ for all i,j. If

$$A_1 = \begin{bmatrix} 0 & 1 & 2 & 5 \\ 1 & 0 & 3 & 3 \\ 2 & 3 & 0 & 2 \\ 4 & 3 & 2 & 0 \end{bmatrix},$$

and

$$A_2 = \begin{bmatrix} 0 & 2 & 3 & 5 \\ 2 & 0 & 5 & 5 \\ 3 & 5 & 0 & 1 \\ 5 & 5 & 1 & 0 \end{bmatrix},$$

then

$$Q_1 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 2 & 1 \\ 0 & 2 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$R_1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

and

$$S_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

3. THE ALGORITHM

Let $\mathbf{x}' = [\mathbf{x}_1', \ldots, \mathbf{x}_F']$. Then the frequency assignment problem takes the general form:

$$\min \quad g(\mathbf{x}) = \mathbf{x}' C \mathbf{x} \tag{4}$$

$$\text{s.t.} \quad \sum_f x_{fn} = 1, \qquad \text{for all } n \tag{5}$$

$$x_{fn} \in \{0,1\}, \qquad \text{for all } f,n \tag{6}$$

where the diagonal elements of C are 0 and all other elements are positive. The continuous relaxation of (4)–(6) is obtained by replacing (6) with

$$0 \le x_{fn} \le 1, \qquad \text{for all } f,n. \tag{7}$$

The model (4), (5), (7) is a nonconvex quadratic program and a local optimum can be efficiently obtained by application of the convex simplex algorithm as described in Zangwill [6]. Suppose we begin with a feasible integer solution $\bar{\mathbf{x}}' = [\bar{\mathbf{x}}_1', \ldots, \bar{\mathbf{x}}_F']$. We assume that all nonbasic variables have a value of zero. Let $l_1, \ldots, l_N$ denote the subscript such that $\bar{x}_{l_1 1} = \cdots = \bar{x}_{l_n n} = 1$. Then a nonbasic variable x_{fn} with a value of zero prices favorably if $[\nabla g(\bar{\mathbf{x}})]'(\mathbf{e}_i - \mathbf{e}_j) < 0$, where $i = (f-1)N + n$ and $j = (f-1)N + l_n$. The line search for this problem requires that we solve the problem

$$\min_{0 \le \Delta \le 1} g(\bar{\mathbf{x}} + (\mathbf{e}_i - \mathbf{e}_j)\Delta). \tag{8}$$

But

$$\begin{aligned} \left.\frac{dg(\bar{\mathbf{x}} + (\mathbf{e}_i - \mathbf{e}_j)\Delta)}{d\Delta}\right|_{\Delta=1} &= (\mathbf{e}_i - \mathbf{e}_j)'\nabla g(\bar{\mathbf{x}} + \mathbf{e}_i - \mathbf{e}_j) \\ &= (\mathbf{e}_i - \mathbf{e}_j)'(C + C')(\bar{\mathbf{x}} + \mathbf{e}_i - \mathbf{e}_j) \\ &= [\nabla g(\bar{\mathbf{x}})]'(\mathbf{e}_i - \mathbf{e}_j) + (\mathbf{e}_i - \mathbf{e}_j)'(C + C')(\mathbf{e}_i - \mathbf{e}_j). \end{aligned}$$

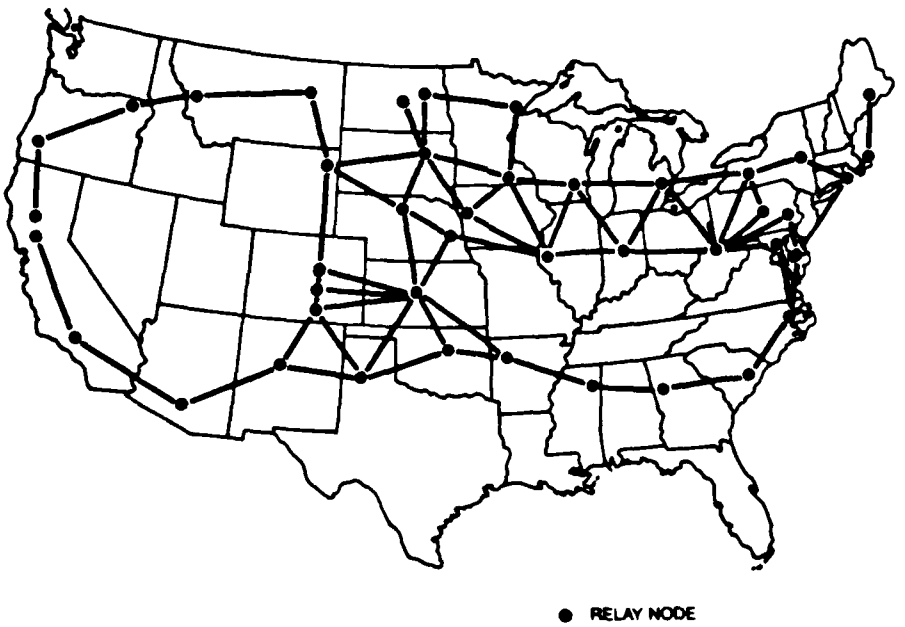


Figure 1. 43-node communication network with designated key links.

Since x_{fn} priced favorably, then $[\nabla g\ (\bar{\mathbf{x}})]'(\mathbf{e}_i - \mathbf{e}_j) < 0$. Also, $(\mathbf{e}_i - \mathbf{e}_j)'(C + C')(\mathbf{e}_i - \mathbf{e}_j) = \mathbf{e}_i'(C + C')\mathbf{e}_i + \mathbf{e}_j'(C + C')\mathbf{e}_j - \mathbf{e}_i'(C + C')\mathbf{e}_j - \mathbf{e}_j'(C + C')\mathbf{e}_i$. But the diagonal elements of $(C + C')$ are 0 and all other elements are non-negative. Hence, the solution to (8) is $\Delta^* = 1$ and the exact change to the objective function will be $\nabla g\ (\bar{\mathbf{x}}')(\mathbf{e}_i - \mathbf{e}_j) - \mathbf{e}_i'(C + C')\mathbf{e}_j - \mathbf{e}_j'(C + C')\mathbf{e}_i$, a strict decrease. Therefore, in the new solution x_{fn} is set to 1 and $x_{l_n n}$ is set to 0. Since this holds for every iteration of the convex simplex algorithm, integrality is maintained and a local optimum for (4)–(6) can be obtained by finding a local optimum for (4), (5), (7).

Let $\bar{\mathbf{x}}$ be any initial assignment for the frequency assignment problem. Using this initial assignment, the algorithm may be stated as follows:

For $f = 1, \ldots, F$.
  For $n = 1, \ldots, N$.
    $l_n := k$, where $\bar{x}_{kn} = 1$.
    $i := (f - 1)N + n$.
    $j := (f - 1)N + l_n$.
    $p := [\nabla g\ (\bar{\mathbf{x}})]'(\mathbf{e}_i - \mathbf{e}_j)$.
    If $p < 0$
      then
        $\bar{x}_{l_n n} := 0$
        $\bar{x}_{fn} := 1$

Repeat as long as $p < 0$ for some f and some n.

4. COMPUTATIONAL EXPERIENCE

We implemented the frequency assignment algorithm in a FORTRAN code. All data, including the matrices Q_f, R_f, and S_f, are stored in high speed core. Special subroutines were written to evaluate both $g(\cdot)$ and $\nabla g\ (\cdot)$ at a point. The code begins with F different starting solutions and stops when a local optimum is found. The initial assignment for run $r \in \{1, \ldots, F\}$ is to assign frequency $\{[(n + r - 2)$ modulo $F] + 1\}$ to node n. The best solution obtained from all F runs is the output.

Table 1. Computational Results With 43 Node Model

	Problem				
Row description	1	2	3	4	5
α *dB*	10	10	12	10	10
β *dB*	25	25	25	25	30
F (channels)	10	12	14	14	14
Binary variables	430	516	602	602	602
Iterations	525	341	497	540	526
Solution time (secs)	5	7	10	11	11
Initial objective value	3153	1561	1875	1671	1670
Final objective value	164	103	79	5	4
Jammed key links	8	4	3	0	0

Five test problems were generated from the real-world 43-node network illustrated in Fig. 1. The lines connecting nodes are the designated key links. The problems all have the same topology but differ in the selection of the critical values and the weighting constants. A random assignment was generated and the matrices were modified so that this assignment produced a cost of zero. Hence, the optimal objective value for each problem is zero.

Our computational experience is reported in Table 1. All runs were made on a CDC Cyber-875 using the FTN5 compiler with OPT = 2. The initial objective value row is the average objective value for the F initial solutions. Note that all five problems were run in less than 1 minute of CPU time and the final objective values were quite close to the optimum as compared to the initial assignments.

5. CONCLUSIONS

Our optimization model and computer software provide a practical approach to assist communication network designers in obtaining near-optimal solutions for the frequency assignment problem. The fact that the diagonal elements of C in the quadratic objective function $\mathbf{x}'C\mathbf{x}$ are zero allows a very efficient implementation of the convex simplex method which maintains integrality. Hence, if we begin with an integer assignment, the convex simplex algorithm follows a sequence of integer points until a local minimum is obtained. This procedure is so fast that very large problems can be easily handled by this approach.

ACKNOWLEDGMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract No. AFOSR 83-0278.

REFERENCES

[1] Collins, M., Cooper, L., Helgason, R., Kennington, J., and LeBlanc, L., "Solving the Pipe Network Analysis Problem Using Optimization Techniques," *Management Science*, **24**(7), 747–760 (1978).

[2] Kennington, J. L., and Helgason, R. V., *Algorithms for Network Programming*, Wiley, New York, 1980.

[3] Kennington, J. L., "A Convex Simplex Code For Solving Nonlinear Network Flow Problems," Department of Operations Research Technical Report No. 82-OR-6, Southern Methodist University, Dallas, TX, 1982.

[4] Mathur, K., Salkin, H., Nishimura, K., and Morito, S., "The Design of an Interactive Computer Software System for the Frequency-Assignment Problem," *IEEE Transactions on Electromagnetic Compatibility*, **EMC-26**(4), 207–212 (1984).

[5] Morito, S., Salkin, H., and Williams, D., "Two Backtrack Algorithms for the Radio Frequency Intermodulation Problem," *Applied Mathematics and Optimization*, **6**, 221–240 (1980).

[6] Zangwill, W. I., *Nonlinear Programming: A Unified Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1969.

[7] Zoellner, J. A., and Beall, C. L., "A Breakthrough in Spectrum Conserving Frequency Assignment Technology," *IEEE Transactions on Electromagnetic Compatibility*, **EMC-19**(3), 313–319 (1977).

Received November 27, 1985
Accepted December 3, 1985

Mathematical Programming 37 (1987) 309-317
North-Holland

CHAPTER 4

A GENERALIZATION OF POLYAK'S CONVERGENCE RESULT FOR SUBGRADIENT OPTIMIZATION

Ellen ALLEN, Richard HELGASON and Jeffery KENNINGTON
Department of Operations Research, Southern Methodist University, Dallas, TX 75275, USA

Bala SHETTY
Department of Business Analysis, Texas A & M University, College Station, TX 77843, USA

Received 20 August 1985
Revised manuscript received 17 November 1986

This paper generalizes a practical convergence result first presented by Polyak. This new result presents a theoretical justification for the step size which has been successfully used in several specialized algorithms which incorporate the subgradient optimization approach.

Key words: Subgradient optimization, nonlinear programming, convergence

1. The subgradient algorithm

Let $G \neq \emptyset$ be a closed and convex subset of R^n. For each $y \in R^n$, define the *projection* of y on G, denoted by $P(y)$, to be the unique point of G such that for all $z \in G$, $\|P(y) - y\| \leq \|z - y\|$. It is well known that the projection exists in this case and that for all $x, y \in R^n$, $\|P(x) - P(y)\| \leq \|x - y\|$.

Let g be a finite convex functional on G. For each $y \in G$, define the *subdifferential* of g at y by

$$\partial g(y) = \{\eta : \eta \in R^n \text{ and for all } z \in G,\ g(z) \geq g(y) + \eta \cdot (z - y)\}.$$

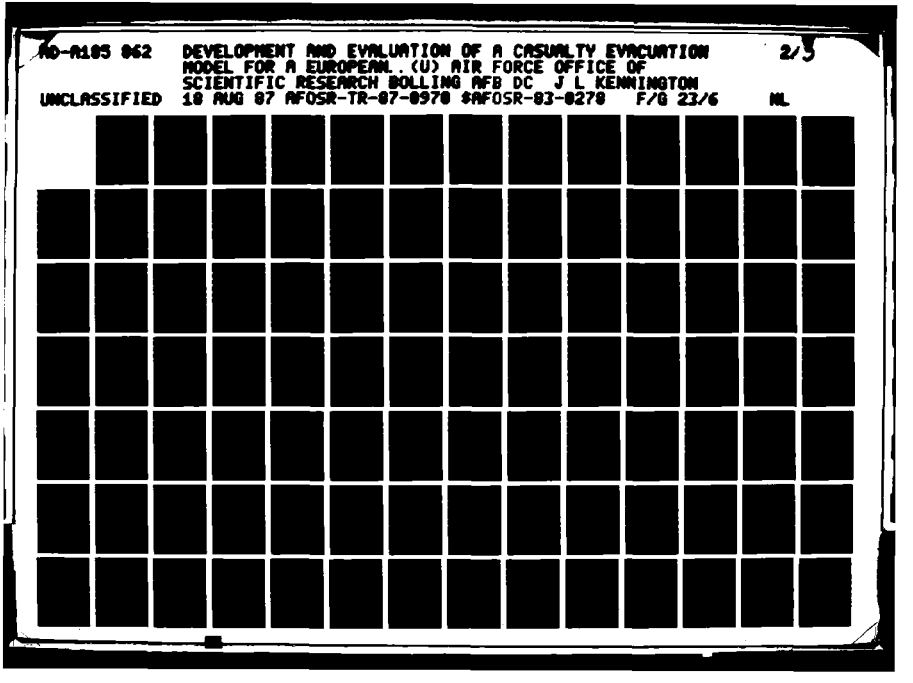
Any $\eta \in \partial g(y)$ is called a *subgradient* of g at y. It is well known that if y is a point at which g is differentiable, then $\partial g(y) = \{\nabla g(y)\}$, a singleton set.

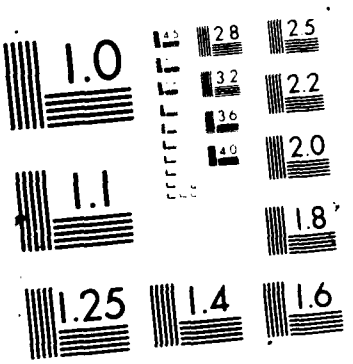
It is also well known that on the relative interior of G, g is continuous and the subdifferential of g always exists. That this may not be the case on the relative boundary is shown in the following simple example.

Example 1. Let $G = [0, 1]$ in R. The finite convex function $f: G \to R$ given by

$$f(y) = \begin{cases} 0, & 0 \leq y < 1, \\ 1, & y = 1, \end{cases}$$

fails to have a subgradient and is discontinuous at the boundary point $y = 1$.


AD-A185 062
DEVELOPMENT AND EVALUATION OF A CASUALTY EVACUATION MODEL FOR A EUROPEAN.. (U) AIR FORCE OFFICE OF SCIENTIFIC RESEARCH BOLLING AFB DC J L KENNINGTON
2/3
UNCLASSIFIED
18 AUG 87 AFOSR-TR-87-0970 $AFOSR-83-0278
F/G 23/6
NL


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

That the notions of continuity and subdifferentiability are independent on the relative boundary of G is shown by the following examples.

Example 2 [21, p. 229]. Let $G=[0,1]$ in R. The finite convex function $f: G \to R$ given by $f(y)=-(1-y)^{1/2}$ is continuous on G but fails to have a subgradient at the boundary point $y=1$.

Example 3 [6, p. 96]. Let $G=\{(y_1, y_2): 0 \leq y_2 \leq y_1 \leq 1\}$ in R^2. The finite (but unbounded) convex functional $f: G \to R$ given by

$$f(y)=\begin{cases}(y_2)^2/y_1, & 0 \leq y_2 \leq y_1,\ 0 < y_1 \leq 1,\\ 0, & y_1 = y_2 = 0,\end{cases}$$

is discontinuous at $(0,0)$ but has a subgradient everywhere on G.

Consider the nonlinear programming problem given by

$$\begin{aligned}&\text{minimize} \quad g(y) && \text{(NLP/SD)}\\ &\text{subject to} \quad y \in G,\end{aligned}$$

where we assume that for all $y \in G$, $\partial g(y) \neq \emptyset$ and that the set of optimal points $\Gamma \neq \emptyset$. We denote the optimal objective value by γ.

The subgradient optimization algorithm for the solution of NLP/SD was first introduced by Shor [23] and may be viewed as a generalization of the steepest descent method in which any subgradient is substituted for the gradient at a point where the gradient does not exist. This algorithm uses a sequence of positive step sizes $\{s_i\}$, which in turn depend on a predetermined sequence of fixed constants $\{\lambda_i\}$ and (in some cases) certain other quantities.

Subgradient optimization algorithm

Step 0 (Initialization)

Let $y_0 \in G$ and set $i \leftarrow 0$.

Step 1 (Find Subgradient and Step Size)

Obtain some $\eta_i \in \partial g(y_i)$.

If $\eta_i = 0$, terminate with y_i optimal; otherwise, select a step size s_i.

Step 2 (Move to New Point)

Set $y_{i+1} \leftarrow P(y_i - s_i \eta_i)$, $i \leftarrow i+1$, and return to step 1.

Unfortunately, the termination criterion in step 1 may not hold at any member of Γ and is thus computationally ineffective. Hence, some other stopping rule must be devised. In practice this is often a limit on the number of iterations. The functional values produced by the algorithm will be denoted by $g_i = g(y_i)$.

Various proposals have been offered for the selection of the step sizes. Four general schema which have been suggested are:

$$s_i = \lambda_i. \tag{1}$$

$$s_i = \lambda_i / \|\eta_i\|, \tag{2}$$

$$s_i = \lambda_i / \|\eta_i\|^2, \tag{3}$$

$$s_i = \lambda_i (g_i - \rho) / \|\eta_i\|^2, \tag{4}$$

where ρ, the *target value*, is an estimate of γ and all $\lambda_i > 0$.

The papers of Polyak [19] and Held, Wolfe and Crowder [12] have provided the major impetus for widespread practical application of the algorithm. Schema (4) has proven to be a particularly popular choice among experimenters. Theorem 4 of Polyak [19] is the most often quoted convergence result justifying use of this schema. For many mathematical programming models, the target value is a lower bound on the optimum (see, e.g., [2, 3, 4, 5, 13, 14, 22]). For this case Polyak's Theorem 4, using schema (4), requires that $\lambda_i = 1$ for all i. For all the above studies, a decreasing sequence of λ's was found to work better than $\lambda_i = 1$ for all i. Hence, the existing theory did not justify what we had found to work well in practice. The objective of this paper is to present improved theoretical results which help to explain what has been found to work well in practice. Specifically, we loosen slightly the restrictions imposed on the sequence $\{\lambda_i\}$, and obtain a more general result when the target value is less than the optimum.

The literature on the subgradient algorithm is extensive, much of it in Russian. Good coverage is contained in the bibliographies of [20], [24], and [25]. Much of this literature has grown up in conjunction with the relaxation method for solving linear inequalities (see, e.g., [1, 7, 10, 11, 15]).

2. Polyak's convergence results

The results of Theorem 4 of Polyak [19] use the following general restrictions on the sequence $\{\lambda_i\}$ used with schema (4):

$$0 < \alpha \leq \lambda_i \leq \beta < 2, \tag{5}$$

where α and β are fixed constants.

The results contained in this theorem include, under (4), (5), and (essentially) the assumption that there is some $\kappa > 0$ such that $\|\eta_i\| < \kappa$:

(A) if $\rho > \gamma$, either

(a) there is some n such that $g_n < \rho$, or

(b) all $g_n \geq \rho$ and $\lim g_n = \rho$;

and

(B) if $\rho < \gamma$ and all $\lambda_n = 1$, given $\delta > 0$,
there is some n such that $g_n \leq \gamma + (\gamma - \rho) + \delta$.

If (b) occurs in part (A), the convergence is geometric. Polyak's theorem contains additional results for $G = R^n$ (so that the projection is superfluous), with all $\lambda_n = 1$ and $\rho \geq \gamma$, in which case geometric or faster convergence to the target value is obtained.

In the next section we will relax condition (5) to the following:

$$0 < \lambda_i \leq \beta < 2 \quad \text{and} \quad \sum \lambda_i = \infty, \tag{6}$$

where β is a fixed constant. With this relaxation (which allows the sequence $\{\lambda_i\}$ to approach zero) we present results analogous to (A) and an interesting generalization of (B).

Stronger convergence results are available for special cases, e.g. where set G contains a set

$$H = \{x: f^i(x) \leq 0, i = 1, \ldots, m\},$$

with each f^i convex and H having a nonempty interior (see, e.g., [8, 16, 17, 18] and [25, Ch. 2]).

3. New convergence results

The main results in this section appear in Propositions 5, 7, 9, and 10. Propositions 5 and 7 correspond to part (A) of Polyak's Theorem 4 with slightly weaker conditions on the sequence $\{\lambda_i\}$ and Proposition 9 is a generalization of part B of Theorem 4. Proposition 10 is a new result apparently obtainable only when the conditions on $\{\lambda_i\}$ are weakened so that we may require $\lambda_i \to 0$.

Proposition 1. *If $y \in G$, then*

$$\|y - y_{i+1}\|^2 \leq \|y - y_i\|^2 + s_i^2 \|\eta_i\|^2 + 2s_i(g(y) - g_i).$$

Proof. Let $y \in G$.

$$\begin{aligned}\|y - y_{i+1}\|^2 &= \|y - P(y_i - s_i\eta_i)\|^2 = \|P(y) - P(y_i - s_i\eta_i)\|^2 \leq \|y - y_i + s_i\eta_i\|^2 \\ &= \|y - y_i\|^2 + s_i^2\|\eta_i\|^2 + 2s_i\eta_i \cdot (y - y_i) \\ &\leq \|y - y_i\|^2 + s_i^2\|\eta_i\|^2 + 2s_i(g(y) - g_i).\end{aligned}$$

Proposition 2. *If $y \in \Gamma$, then under* (4),

$$\|y - y_{i+1}\|^2 \leq \|y - y_i\|^2 + \lambda_i(g_i - \rho)[\lambda_i(g_i - \rho) - 2(g_i - \gamma)]/\|\eta_i\|^2.$$

Proof. Let $y \in \Gamma$. Substituting in Proposition 1 for s_i from (4) and using $g(y) = \gamma$, we obtain

$$\begin{aligned}\|y - y_{i+1}\|^2 &\leq \|y - y_i\|^2 + \lambda_i^2(g_i - \rho)^2/\|\eta_i\|^2 + 2\lambda_i(g_i - \rho)(\gamma - g_i)/\|\eta_i\|^2 \\ &= \|y - y_i\|^2 + \lambda_i(g_i - \rho)[\lambda_i(g_i - \rho) - 2(g_i - \gamma)]/\|\eta_i\|^2.\end{aligned}$$

Proposition 3. *If $y \in \Gamma$, $\rho \geq \gamma$, and $g_i \geq \rho$, then under* (4),

$$\|y - y_{i+1}\|^2 \leq \|y - y_i\|^2 + \lambda_i(\lambda_i - 2)(g_i - \rho)^2/\|\eta_i\|^2.$$

Proof. Let $y \in \Gamma$, $\rho \geq \gamma$, and $g_i \geq \rho$. Now

$$\lambda_i(g_i - \rho) - 2(g_i - \gamma) \leq \lambda_i(g_i - \rho) - 2(g_i - \rho) = (\lambda_i - 2)(g_i - \rho).$$

Thus,

$$\lambda_i(g_i - \rho)[\lambda_i(g_i - \rho) - 2(g_i - \gamma)]/\|\eta_i\|^2 \leq \lambda_i(\lambda_i - 2)(g_i - \rho)^2/\|\eta_i\|^2.$$

The result now follows from Proposition 2.

Proposition 4. *If $y \in \Gamma$, $\rho \geq \gamma$, and all $g_i > \rho$ unde* (4) *with all $\lambda_i < 2$, then the sequence $\{\|y - y_i\|^2\}$ is monotone decreasing and converges to some $\psi \geq 0$.*

Proof. Let $y \in \Gamma$, $\rho \geq \gamma$, all $\lambda_i < 2$, and all $g_i \geq \rho$. Since each $\lambda_i < 2$, then also each $\lambda_i(\lambda_i - 2)(g_i - \rho)^2/\|\eta_i\| < 0$, and from Proposition 3, $\{\|y - y_i\|^2\}$ is a monotone decreasing sequence. This sequence is bounded below by zero and thus converges to some nonnegative value, say ψ.

Proposition 5. *If $\rho \geq \gamma$ and there is some $\kappa > 0$ such that all $\|\eta_i\| < \kappa$, then under* (4) *and* (6), *given $\delta > 0$, there is some M such that $g_M \leq \rho + \delta$.*

Proof. Let $\delta > 0$ be given, with $\rho \geq \gamma$, and all $\|\eta_i\| < \kappa$. Suppose, contrary to the desired result, that all $g_i > \rho + \delta$. Take any $y \in \Gamma$. Then from Proposition 3,

$$\lambda_i(2 - \lambda_i)(g_i - \rho)^2/\|\eta_i\|^2 \leq \|y - y_i\|^2 - \|y - y_{i+1}\|^2.$$

Since $\lambda_i \leq \beta < 2$, $\|\eta_i\| < \kappa$, and $g_i - \rho > \delta$,

$$\lambda_i(2 - \beta)\delta^2/\kappa^2 \leq \|y - y_i\|^2 - \|y - y_{i+1}\|^2. \tag{7}$$

Adding together the inequalities obtained from (7) by letting i take on all values from 0 to n, we obtain

$$(\lambda_0 + \cdots + \lambda_n)(2 - \beta)\delta^2/\kappa^2 \leq \|y - y_0\|^2 - \|y - y_{n+1}\|^2. \tag{8}$$

As n goes to ∞, the left side of (8) goes to ∞, whereas, by Proposition 4, the right side of (8) goes to $\|y - y_0\|^2 - \psi^2$, a contradiction.

Proposition 5 gives a practical convergence result when the target exceeds the optimal value. At worst we eventually obtain an objective value arbitrarily close to the target value. At best we may obtain an objective value as good as or better than the target value, in which case it may be desirable to restart the algorithm after supplying a new target value.

It does appear theoretically possible that no iterate may have an objective value as good as or better than the target value. In this case, we obtain convergence results in Proposition 7 analogous to Polyak's Theorem 4, part (A), alternative (b).

Proposition 6. *If $\rho \geq \gamma$ and all $g_i > \rho$ under* (4) *and* (6), *then the sequence $\{y_i\}$ is bounded.*

Proof. Let $y \in \Gamma$, $\rho \geq \gamma$, and all $g_i > \rho$. Now,

$$\|y_i\| = \|y_i - y + y\| \leq \|y_i - y\| + \|y\|.$$

From Proposition 4, we then have that

$$\|y_i\| \leq \|y_0 - y\| + \|y\|.$$

Proposition 7. *If $\rho \geq \gamma$, there is some $\kappa > 0$ such that all $\|\eta_i\| < \kappa$, and all $g_i > \rho$ under* (4) *and* (6), *then $\{y_i\}$ converges to some $z \in G$ and some subsequence $\{g_{N(j)}\}$ converges to $\rho \geq g(z)$. If g is also continuous on G, then $\{g_i\}$ converges to $\rho = g(z)$.*

Proof. Let $\rho \geq \gamma$, all $g_i > \rho$, and all $\|\eta_i\| < \kappa$. Using Proposition 5, we obtain a convergent subsequence of $\{g_i\}$. There is some $M(0)$ such that $g_{M(0)} \leq \rho + 1$. Having determined $M(j)$, define $h_j = \min\{g_0, \ldots, g_{M(j)}\} - \rho > 0$ and $\delta_j = \min\{(\frac{1}{2})^{j+1}, h_j/2\}$. Applying Proposition 5, there is some $M(j+1)$ such that $g_{M(j+1)} \leq \rho + \delta_j$ and furthermore, $M(j+1) > M(j)$. As constructed, $\{g_{M(j)}\}$ converges to ρ. By Proposition 6, $\{y_{M(j)}\}$ is bounded, so a subsequence of $\{y_{M(j)}\}$, say $\{y_{N(j)}\}$, converges to some point z. Obviously, $\{g_{N(j)}\}$ also converges to ρ. Since G is closed, $z \in G$. Let η be any subgradient of g at z. Thus for each j, $g_{N(j)} \geq g(z) + \eta(y_{N(j)} - z)$, from which it follows that $\rho \geq g(z)$. Now consider the ancillary functional $g^\sigma : G \to R$, given by $g^\sigma(y) = \max\{g(y), \sigma\}$. Note that g^σ is convex and finite on G and z is a minimum point of g^σ over G. Also, since $g^\sigma(y) \geq g(y)$ and each $g_i > \rho$, each subgradient η_i of g at y_i is also a subgradient of g^σ at y_i. Thus under (4) and (6), for each i, $g_i = g_i^\sigma$. By Proposition 4, applied to g^σ, $\{y_i\}$ converges to z. When g is continuous on G, $\lim g_i = g(z) = \lim g_{N(j)} = \rho$.

If we simply require that subgradients exist for all points generated by the subgradient optimization algorithms, and relax the requirement that subgradients exist at all points on the relative boundary of G, then, contrary to the result in Proposition 7, when all $g_i > \rho$, $\{y_i\}$ can converge to a point z with $g(z) > \rho$, as shown in the following example.

Example 4. Let $G = \{(y_1, y_2): 0 \leq y_1, y_2 \leq 1\}$ in R^2. The finite convex functional $f: G \to R$ given by

$$f(y) = \begin{cases} (y_1 - 1)^2, & (y_1, y_2) \neq (1, 1), \\ 1, & (y_1, y_2) = (1, 1), \end{cases}$$

fails to have a subgradient at the boundary point $z = (1, 1)$. Also, $\gamma = 0$ with $\Gamma = \{(y_1, y_2): y_1 = 1, 0 \leq y_2 < 1\}$. Letting $\lambda_i = 1$, for all i, (6) holds. Then under (4), starting from $y_0 = (0, 1)$ with $\rho = 0 = \gamma$, the subgradient optimization algorithm generates the points $y_i = (1 - (\frac{1}{2})^i, 1)$ with $g_i = (\frac{1}{4})^i$. Now $\lim y_i = z$, but $\lim g_i = 0 = \rho < g(z) = 1$.

Proposition 8. *If $y \in \Gamma$, $g_i \geq \rho$, and $\lambda_i \leq \beta \neq 2$, then under* (4),

$$\|y - y_{i+1}\|^2 \leq \|y - y_i\|^2 + \lambda_i(g_i - \rho)(2-\beta)[(\gamma - g_i) + (\beta/(2-\beta))(\gamma - \rho)]/\|\eta_i\|^2.$$

Proof. Let $y \in \Gamma$, $g_i \geq \rho$, and $\lambda_i \leq \beta \neq 2$. Now,

$$\begin{aligned}\lambda_i(g_i - \rho) - 2(g_i - \gamma) &\leq \beta(g_i - \rho) - 2(g_i - \gamma)\\ &= \beta(g_i - \rho) - (2-\beta)(g_i - \gamma) - \beta(g_i - \gamma)\\ &= \beta(\gamma - \rho) + (2-\beta)(\gamma - g_i)\\ &= (2-\beta)[(\gamma - g_i) + (\beta/(2-\beta))(\gamma - \rho)].\end{aligned}$$

Thus,

$$\begin{aligned}&\lambda_i(g_i - \rho)[\lambda_i(g_i - \rho) - 2(g_i - \gamma)]/\|\eta_i\|^2\\ &\quad \leq \lambda_i(g_i - \rho)(2-\beta)[(\gamma - g_i) + (\beta/(2-\beta))(\gamma - \rho)]/\|\eta_i\|^2.\end{aligned}$$

The result now follows from Proposition 2.

Proposition 9. *If $\rho < \gamma$ and there is some $\kappa > 0$ such that all $\|\eta_i\| < \kappa$, then under* (4) *and* (6), *given $\delta > 0$, there is some M such that $g_M \leq \gamma + (\beta/(2-\beta))(\gamma - \rho) + \delta$.*

Proof. Let $\delta > 0$ be given, with $\rho < \gamma$, and all $\|\eta_i\| < \kappa$. Suppose, contrary to the desired result, that all $g_i > \gamma + (\beta/(2-\beta))(\gamma - \rho) + \delta$, or $(\gamma - g_i) + (\beta/(2-\beta)) \times (\gamma - \rho) < -\delta$. Since $\beta < 2$ and $g_i > \rho$, then

$$\begin{aligned}&\lambda_i(g_i - \rho)(2-\beta)[(\gamma - g_i) + (\beta/(2-\beta))(\gamma - \rho)]/\|\eta_i\|^2\\ &\quad < -\delta\lambda_i(g_i - \rho)(2-\beta)/\|\eta_i\|^2. \qquad (9)\end{aligned}$$

Take any $y \in \Gamma$. Then by (9) and Proposition 8, we have that

$$\delta\lambda_i(g_i - \rho)(2-\beta)/\|\eta_i\|^2 < \|y - y_i\|^2 - \|y - y_{i+1}\|^2.$$

Since $\|\eta_i\| < \kappa$ and $g_i \geq \gamma > \rho$, then also

$$\lambda_i\delta(\gamma - \rho)(2-\beta)/\kappa^2 < \|y - y_i\|^2 - \|y - y_{i+1}\|^2. \qquad (10)$$

Adding together the inequalities obtained from (10) by letting i take on all values from 0 to n, we obtain

$$(\lambda_0 + \cdots + \lambda_n)\delta(\gamma - \rho)(2-\beta)/\kappa^2 < \|y - y_0\|^2 - \|y - y_{n+1}\|^2. \qquad (11)$$

As n goes to ∞, the left side of (11) goes to ∞, whereas, by Proposition 4, the right side of (11) goes to $\|y - y_0\|^2 - \psi^2$, a contradiction.

The above is our generalization of Polyak's Theorem 4 Part (B). At worst we eventually obtain an objective value whose error is arbitrarily close to $\beta/(2-\beta)$ times the error present in the target value estimate of γ.

Proposition 10. *If $\rho < \gamma$, there is some $\kappa > 0$ such that all $\|\eta_i\| < \kappa$, and all $g_i > \gamma$ under (4) and (6) with $\lambda_i \to 0$, then there is a subsequence $\{g_{M(j)}\}$ which converges to γ.*

Proof. Let $\rho < \gamma$, all $\|\eta_i\| < \kappa$, $\lambda_i \to 0$, and all $g_i > \gamma$. Using Proposition 9, we obtain a convergent subsequence of $\{g_i\}$. Define $\beta_0 = \min\{1, 1/(\gamma - \rho)\}$ and $\delta_0 = 1$. There is some $N(0)$ such that for all $i \geq N(0)$, $\lambda_i < \beta_0$. Then also $(\beta_0/(2-\beta_0))(\gamma - \rho) \leq 1$. Applying Proposition 9 to $\{g_{i-N(0)}\}$, there is some $M(0) \geq N(0)$ such that $g_{M(0)} \leq \gamma + (\beta_0)/(2-\beta_0)(\gamma - \rho) + \delta_0 \leq \gamma + 2$. Having determined $M(j)$, define $h_j = \min\{g_0, \ldots, g_{M(j)}\} - \gamma > 0$, $\delta_{j+1} = \min\{(\frac{1}{2})^{j+1}, h_j/3\}$, and $\beta_{j+1} = \min\{1, \delta_{j+1}/(\gamma - \rho)\}$. There is some $N(j+1)$ such that for all $i \geq N(j+1)$, $\gamma_i < \beta_{j+1}$. Then also $(\beta_{j+1}/(2-\beta_{j+1}))(\gamma - \rho) < \delta_{j+1}$. Applying Proposition 9 to $\{g_{i-N(j+1)}\}$, there is some $M(j+1) \geq N(j+1)$ such that

$$g_{M(j+1)} \leq \gamma + (\beta_{j+1}/(2-\beta_{j+1}))(\gamma - \rho) + \delta_{j+1} \leq \gamma + 2\delta_{j+1}.$$

Then also

$$g_{M(j+1)} \leq \gamma + (\tfrac{1}{2})^j \quad \text{and} \quad g_{M(j+1)} \leq \gamma + (2/3)h_j < \min\{g_0, \ldots, g_{M(j)}\},$$

so that $M(j+1) > M(j)$. As constructed, $\{g_{M(j)}\}$ converges to γ.

4. Conclusions

Propositions 5 and 7 give the convergence results obtained under (4) and (6) for a target value at or above the optimal value. It is readily apparent that Proposition 5 is compatible with Polyak's result (A). Proposition 9 gives the corresponding result for a target value under the optimal value. We have found this to be a more practical result (see, e..g., [2, 3, 4, 5, 13, 14, 22]). Taking $\beta = 1$, we have Polyak's result (B) as a special case of Proposition 9. Proposition 9 shows more clearly the dependence of the demonstrably attainable error on the upper bound β for $\{\lambda_i\}$. Requiring $\lambda_i \to 0$ allows us to produce a subsequence of objective values converging to the optimal objective as shown in Proposition 10. This paper has not addressed the question of any convergence rate associated with the use of (4) and (6). Goffin [9] has provided such results when schema (2) is used.

Acknowledgment

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR 83-0278. We wish to thank David Anderson of the Department of Mathematics at Southern Methodist University for helpful suggestions concerning the form of Proposition 9. We are deeply indebted to referee Jean-Louis Goffin of McGill University, who has provided extensive assistance and valuable suggestions for improvements in this paper, especially Propositions 7 and 10. He has also furnished key elements in the proof of Proposition 7.

References

[1] S. Agmon, "The relaxation method for linear inequalities," *Canadian Journal of Mathematics* 6 (1954) 382-392.

[2] I. Ali, "Two node-routing problems," unpublished dissertation, Department of Operations Research, Southern Methodist University (Dallas, Texas, 1980).

[3] I. Ali and J. Kennington, "The asymmetric *M*-travelling salesman problem: A duality based branch-and-bound algorithm," *Discrete Applied Mathematics* 13 (1986) 259-276.

[4] I. Ali, J. Kennington and B. Shetty, "The equal flow problem," Technical Report 85-OR-1, Operations Research Department, Southern Methodist University, Dallas, Texas (1980).

[5] E. Allen, "Using two sequences of pure network problems to solve the multicommodity network flow problem," unpublished dissertation, Department of Operations Research, Southern Methodist University, Dallas, Texas (1985).

[6] M. Bazaraa and C.M. Shetty, *Foundations of Optimization*, Lecture Notes in Economics and Mathematical Systems No. 122 (Springer-Verlag, Berlin, 1976).

[7] I. I. Eremin, "An iterative method for Cebysev approximations of incompatible systems of linear inequalities," *Soviet Mathematics Doklady* 3 (1962) 570-572.

[8] I.I. Eremin, "On systems of inequalities with convex functions in the left sides," *American Mathematical Society Translations*, 88(2) (1970) 67-83.

[9] J.L. Goffin, "On convergence rates of subgradient optimization methods," *Mathematical Programming* 13 (1977) 329-347.

[10] J.L. Goffin, "Nondifferentiable optimization and the relaxation method," in: C. Lemarechal and R. Mifflin, eds., *Nonsmooth Optimization*, IIASA Proceedings Series (Pergamon Press, Oxford, 1978) 31-49.

[11] J.L. Goffin, "Acceleration in the relaxation method for linear inequalities and subgradient optimization," in: E.A. Numrinski, ed., *Progress in Nondifferentiable Optimization* (IIASA, Laxenburg, Austria, 1982) 29-59.

[12] M. Held, P. Wolfe and H. Crowder, "Validation of subgradient optimization," *Mathematical Programming* 6 (1974) 66-68.

[13] R. Helgason, "A Lagrangean relaxation approach to the generalized fixed charge multicommodity minimal cost network flow problem," unpublished dissertation, Department of Operations Research, Southern Methodist University, Dallas, Texas (1980).

[14] J. Kennington and M. Shalaby, "An effective subgradient procedure for minimal cost multicommodity flow problems," *Management Science* 23(9) (1977) 994-1004.

[15] T. Motzkin and I.J. Schoenberg, "The relaxation method for linear inequalities," *Candian Journal of Mathematics* 6 (1954) 393-404.

[16] E.A. Nurminskii, "Convergence conditions for nonlinear programming algorithms," *Cybernetics*, 8(6) (1972) 959-962.

[17] E.A. Nurminskii, "The quasigradient method for solving of the nonlinear programming problem," *Cybernetics* 9(1) (1973) 145-150.

[18] B.T. Polyak, "A general method of solving extremum problems," *Soviet Mathematics Doklady* 8 (1967) 593-597.

[19] B.T. Polyak, "Minimization of unsmooth functionals," *USSR Computational Mathematics and Mathematical Physics*, 9(3) (1969) 14-29.

[20] B.T. Polyak "Subgradient methods: A survey of Soviet research," in: C. Lemarechal and R. Mifflin, eds. *Nonsmooth Optimization*, IIASA Proceedings Series (Pergamon Press, Oxford, 1978) 5-29.

[21] R.T. Rockafellar, *Convex Analysis* (Princeton University Press, Princeton, NJ, 1970).

[22] B. Shetty, "The equal flow problem," unpublished dissertation, Department of Operations Research, Southern Methodist University, Dallas, Texas (1985).

[23] N.Z. Shor, "On the structure of algorithms for the numerical solution of optimal planning and design problems," dissertation, Cybernetics Institute, Academy of Science, USSR (1964).

[24] N.Z. Shor, "Generalization of gradient methods for non-smooth functions and their applications to mathematical programming," *Economics and Mathematical Methods* (in Russian) 12(2) (1976) 337-356.

[25] N.Z. Shor, *Minimization Methods for Non-Differentiable Functions* (Springer-Verlag, Berlin, 1985)

CHAPTER 5

The Equal Flow Problem

Agha Iqbal Ali
University of Texas at Austin

Jeff Kennington
Southern Methodist University

Bala Shetty
Texas A&M University

May 1987

Abstract. This paper presents a new algorithm for the solution of a network problem with equal flow side constraints. The solution technique is motivated by the desire to exploit the special structure of the side constraints and to maintain as much of the characteristics of pure network problems as possible. The proposed algorithm makes use of Lagrangean relaxation to obtain a lower bound and decomposition by right-hand-side allocation to obtain upper bounds. The Lagrangean dual serves not only to provide a lower bound used to assist in termination criteria for the upper bound, but also allows an initial allocation of equal flows for the upper bound. The algorithm has been tested on problems with up to 1500 nodes and 6000 arcs. Computational experience indicates that solutions whose objective function value is well within 1% of the optimum can be obtained in 1%-65% of the MPSX time depending on the amount of imbalance inherent in the problem. Incumbent integer solutions which are within 99.99% feasible and well within 1% of the proven lower bound are obtained in a straightforward manner requiring, on the average, 30% of the MPSX time required to obtain a linear optimum.

Keywords. Networks, Algorithms, Computational Mathematical Programming, Decomposition, Lagrangean Relaxation

Acknowledgement. This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR 83-0278, the Department of Defense under Contract Number MDA 903-86-C-0182, and Rome Air Development Center under Contract Number SCEEE PDP/86-75.

1 Introduction

This paper makes use of relaxation in conjunction with decompostion for the solution of the equal flow problem. The problem is easily conceptualized as a minimal cost network flow problem with additional constraints on certain pairs of arcs. Specifically, given pairs of arcs are required to take on the same value. The problem is defined on a network represented by an m x n node-arc incidence matrix, **A**, in which K pairs of arcs are identified and required to have equal flow. Mathematically, this is expressed as:

$$
\begin{aligned}
&\text{Minimize } \mathbf{cx} \\
&\text{s.t.} \quad \mathbf{Ax} = \mathbf{b} \\
&\qquad x_k = x_{k+K}, \quad k = 1,2,\ldots,K \\
&\qquad \mathbf{0} \le \mathbf{x} \le \mathbf{u} \\
&\qquad \mathbf{x}, \text{ integer}
\end{aligned}
$$

where **c** is a 1 x n vector of unit costs, **b** is an m x 1 vector of node requirements, **0** is an n x 1 vector of zeroes, **x** is an n x 1 vector of decision variables, and **u** is an n x 1 vector of upper bounds. This mathematical statement of the problem, henceforth referred to as problem I1, assumes that the first 2K arcs appear in the equal flow constraints. This is not a restrictive assumption, since by rearranging the order of the arcs, any equal flow problem with K pairs can be expressed in the above form. Note that the K pairs of arcs are mutually exclusive, i. e., an arc appears in at most one side constraint. We also assume without loss of generality, that $u_k = u_{k+K}$ for $k = 1,2,\ldots,K$.

Applications of the equal flow problem include crew scheduling [5], estimating driver costs for transit operations [14], and the two duty period scheduling problem [11]. When integrality constraints are not present, the model is referred to as the linear equal flow problem (P1). The linear model is applicable to problems where integrality is not restrictive. For example, in federal matching of funds allocated to various projects [4]. The linear equal flow problem may be solved using a specialization of the simplex method for networks with side constaints [3]. It has also been solved by transformation to a nonlinear programming problem [4].

The use of relaxation techniques and/or decomposition techniques in the solution of problems with special structure in the constraint set is motivated by potential computational efficiencies. Glover, Glover and Martinson [6] address a generalized network problem in which arcs in specified subsets must have proportional flow. The solution approach is via solution of a series of problem relaxations and progressive bound ad-

justment. The underlying principle is shared in the ensuing development for the equal flow problem.

Lagrangean relaxation has been used to aid in the solution of the integer equal flow problem in two specific instances. Shepardson and Marsten [11] reformulate the two duty period scheduling problem as a single duty period scheduling problem with equal flow side constraints and integrality constraints on the variables. Turnquist and Malandraki [14] model the problem of estimating driver costs for transit operations as an integer equal flow problem. In both studies, the side constraints are dualized and the Lagrangean dual solved using subgradient optimization to yield a lower bound on the optimal objective value. In [14] step-size determination during the subgradient optimization process is aided by a line search.

The objective of this investigation is to develop and computationally test a new algorithm, based on relaxation and decomposition, for the linear equal flow problem and its use in solving the integer equal flow problem. The linear equal flow problem is a natural relaxation for the integer problem and also provides an approximation to the integer model. Because the problems are very closely allied, primarily due to the unimodularity of the node-arc incidence matrix, solutions to the integer model can be obtained by using a slight modification of the technique for the linear model. By employing relaxation and decomposition, solution of the equal flow problem is via two sequences of pure network problems, totally eliminating the computational overhead associated with maintaining the inverse of a basis matrix. The exploitation of the special structure of the side constraints and the network structure results in a decrease in both computer storage and computation time since reoptimization procedures are applicable for solution of subproblems of the two sequences.

The solution technique consists of making use of the Lagrangean dual of the equal flow problem with the side constraints dualized to obtain a lower bound. The Lagrangean relaxation of the equal flow problem does not enforce the equal flow constraints. The Lagrangean dual for the linear and the integer equal flow problem is exactly the same, since the constraint set for the Lagrangean relaxation is identical. This Lagrangean dual is similar to the quadratic programming problem used in [4]. The similarity lies in penalizing the violating equal flow constraints.

Upper bounds are obtained by use of a decomposition of the equal flow problem based on parametric changes in the requirements vector. The Lagrangean dual provides a lower bound which is used to aid the the solution of the decomposition model in determining an initial right-hand-side allocation as well as providing a lower bound on the objective so that a solution is known to be within a percentage of the optimal. As such,

the algorithm can terminate when a solution with a prespecified tolerance on the objective function value is obtained. By enforcing that the parametric changes in the requirements vector be such that integral allocations of equal flow be obtained, upper bounds on the integer problem can be obtained.

The solution technique makes use of subgradient optimization in the solution of both the Lagrangean dual for obtaining a lower bound and the decomposition model for obtaining the upper bound. Both the lower and upper bounding algorithms have been developed in the context of the general subgradient algorithm which is briefly presented in Section 2. Section 3 introduces the Lagrangean dual for the equal flow problem and the lower bounding algorithm. Section 4 presents the decomposition of the linear equal flow problem and the upper bounding algorithm. The overall procedure which makes use of the algorithms of Sections 3 and 4 is given in Section 5, computational results are given in Section 6 and conclusions drawn in Section 7.

2 The Subgradient Algorithm

The subgradient algorithm was first introduced by Shor [13] and provides a framework for solving nonlinear programming problems. It may be viewed as a generalization of the steepest descent (ascent) method for convex (concave) problems in which the gradient may not exist at all points. At points at which the gradient does not exist, the direction of movement is given by a subgradient. Subgradients do not necessarily provide improving directions and consequently, the convergence results of Zangwill [15] do not apply. Convergence of the subgradient algorithm is assured, however, under fairly minor conditions on the step size.

Given the nonlinear program P0,

$$\begin{aligned} &\text{Minimize} \quad f(\mathbf{y}) \\ &\text{s.t.} \qquad \mathbf{y} \in \mathbf{G} \end{aligned}$$

where f is a real-valued function that is convex over the compact, convex, and nonempty set **G**, a vector η is a *subgradient* of f at $\mathbf{y}'$ if $f(\mathbf{y}) - f(\mathbf{y}') \geq \eta(\mathbf{y} - \mathbf{y}')$ for all $\mathbf{y} \in \mathbf{G}$. For any given $\mathbf{y}' \in \mathbf{G}$, the set of all subgradients of f at $\mathbf{y}'$ is denoted by $\partial f(\mathbf{y}')$. Moving a sufficiently large distance s along $-\eta$ can yield a point $\mathbf{x} = \mathbf{y}' - s\eta$ such that $\mathbf{x} \notin \mathbf{G}$. The projection of the point **x** onto **G**, denoted by $P[\mathbf{x}]$, is defined to be the unique point $\mathbf{y} \in$

G that is nearest to x with respect to the Euclidean norm. Using the projection operation, the subgradient algorithm in its most general form follows:

ALGORITHM 1: SUBGRADIENT OPTMIZATION ALGORITHM

0 Initialization

Let $y^0 \in G$,

Select a set of step sizes $s_0, s_1, s_2, \ldots$

$i \leftarrow 0$.

1 Find Subgradient

Let $\eta_i \in \partial f(y^i)$.

If $\eta_i = 0$, then terminate with y^i optimal.

2 Move to new point

$y^{i+1} \leftarrow P[y^i - s_i\eta_i]$

$i \leftarrow i + 1$, and return to step 1.

There are three general schema which can be used in determining the step size when the subgradient algorithm is implemented for a specific problem:

i. $s_i = \lambda_i$

ii. $s_i = \lambda_i / \|\eta_i\|^2$

iii. $s_i = \lambda_i (f(y^i) - F) / \|\eta_i\|^2$

where F is an estimate of f^*, the optimal value of f over G. A summary of the known convergence results for this algorithm may be found in [2] and [10].

3 The Lower Bound

A lower bound on the objective function of the equal flow problem, I1 or P1, can be obtained by using the Lagrangean dual of the problem. The lower bound is used in the step size determination, termination criteria, and determination of an initial equal flow allocation for the upper bound procedure. Associating the Lagrange multiplier w_k with the kth equal flow constraint and defining the K-vector $w = (w_1, w_2, \ldots, w_K)$, the Lagrangean dual for P1, referred to as problem D1, may be stated as

maximize h(w)

$w \in R^K$

where $h(w) = \min\{cx + \Sigma_k w_k(x_k - x_{K+k}) \mid Ax = b, 0 \le x \le u\}$. Since P1 is a linear program, it is easily established that the optimal objective values of P1 and D1 are equal and that any feasible solution to D1 provides a lower bound on the optimal objective value for problems P1 and I1. For any given value of the vector w, the Lagrangean relaxation is a pure network problem. The subgradient of h at a point $\bar{w}$ is given by the K-vector

$$\mathbf{d} = (\bar{x}_1 - \bar{x}_{K+1}, \ldots, \bar{x}_K - \bar{x}_{2K})$$

where $\bar{x}$ solves the Lagrangean relaxation at $\bar{w}$, given by

$$\{\min cx + \Sigma_k \bar{w}_k(x_k - x_{K+k}) \mid Ax = b, 0 \le x \le u\}.$$

ALGORITHM 1 assumed the function f(y) to be convex, whereas h(w) is piecewise linear concave. The lower bounding algorithm, ALGORITHM 2, modifies the framework of the previous algorithm for a concave function. The step sizes used are given by $\lambda_0 = p$, and successive values of λ_i depend on the progressive improvement of the objective and a parameter m*. As long as the objective function continues to improve across m* iterations, the same value of the multiplier is retained. If the objective does not improve over m* iterations, the multiplier is halved, and successive iterations continue from the point where the incumbent best objective function value is found for the previous value of the multiplier. The algorithm makes use of a scalar, UBND, representing an upper bound for the problem. Since the solution procedure progressively improves both the lower bound and the upper bound for the equal flow problem, each time the lower bound algorithm is invoked the value for UBND is obtained from the upper bound procedure. For this algorithm, we assume that both bounds are greater than zero.

Several termination criteria are pertinent to the lower bound algorithm. If the value of the multiplier becomes negligibly small, further improvement in the lower bound is negligible. Such termination criteria are relevant particularly to the initial invocation of the lower bound algorithm since no valid estimate of the upper bound is available. Further, the maximum number of iterations allowed in the initial invocation of the lower bound procedure should be chosen to be larger than in subsequent invocations.

ALGORITHM 2: LOWER BOUND ALGORITHM

1 Initialization

Initialize UBND, step size p, m^*, and tolerance ε.

$w \leftarrow 0$, $m' \leftarrow 0$, $d' \leftarrow \infty$, $I \leftarrow 0$.

2 Find Subgradient

$I \leftarrow I + 1$.

Let $\bar{x}$ solve $h(w) = \min\{cx + \Sigma_k w_k(x_k - x_{K+k}) \mid Ax = b, 0 \leq x \leq u\}$.

$d \leftarrow (\bar{x}_1 - \bar{x}_{K+1}, \ldots, \bar{x}_K - \bar{x}_{2K})$.

If $\|d\| < \|d'\|$, $d' \leftarrow d$, $x' \leftarrow \bar{x}$

If $d = 0$, terminate.

If $h(w) <$ LBND,

$m' \leftarrow m' + 1$,

if $m' = m^*$ $p \leftarrow p/2$, $w \leftarrow w^*$, $d \leftarrow d^*$, $m' \leftarrow 0$;

otherwise,

$m' \leftarrow 0$,

LBND $\leftarrow h(w)$.

$w^* \leftarrow w$

$d^* \leftarrow d$

If (UBND -LBND) $\leq \varepsilon$(UBND), terminate.

3 Move to new point

(a) $w \leftarrow w + pd$.

(b) If max{pd,} < .005, terminate.

(c) Go to step 2 .

The choice of the initial value of p should be directed by the range of objective function coefficients for the problem as well as an estimate of the elements of the vector **d**. This choice can be made automatically when the Lagrangean relaxation is solved with $w = 0$. Since it is the elements of **d** which cause the objective function coefficients to change in each successive iteration of the subgradient optimization procedure, an initial value of p which keeps objective function coefficients from taking on values far away from the original range is a prudent choice. Note that termination of the lower bound procedure can occur when further changes in objective function coefficients is minimal as in step 3(b).

4 The Upper Bound

An alternate formulation of problem P1, referred to as P2, obtained by decomposing the problem is given by

$$\begin{aligned} &\text{Minimize} \quad g(y) \\ &\text{s.t.} \quad\quad y \in \mathbf{S} \end{aligned}$$

where for any vector $y = (y_1, y_2, \ldots, y_K)$,

$$g(y) = \{\min cx | Ax = b;\ 0 \le x \le u;\ x_k = x_{K+k} = y_k,\ k = 1,2,..,K\},$$

and,

$$\mathbf{S} = \{y |\ 0 \le y_k \le u_k, \text{ for } k = 1,2,...,K\}.$$

The decomposition assures the satisfaction of the equal flow constraints. The decomposed problem P2 is equivalent to the problem P1 [12] and may be solved using a specialization of the subgradient optimization algorithm. The objective function is piece-wise linear convex and the subgradient η of g at a point y is obtained from the dual variables, v_i, $i = 1,2,...,2K$, associated with the equal flow constraints in the subproblem, referred to as P3 and given by,

$$\begin{aligned} &\text{Minimize} && cx && && \\ &\text{s.t.} && Ax && = b && \\ &&& x_1 && = y_1 && (v_1) \\ &&& x_{K+1} && = y_1 && (v_{K+1}) \\ &&& \vdots && \vdots && \vdots \\ &&& x_K && = y_K && (v_K) \\ &&& x_{2K} && = y_K && (v_{2K}) \\ &&& 0 \le x \le u. && && \end{aligned}$$

The K-vector

$$\eta = (v_1 + v_{K+1}, v_2 + v_{K+2}, \ldots, v_K + v_{2K})$$

is a *subgradient* of g at $y = (y_1, y_2, \ldots, y_K)$.

The dual variables v_k, $k = 1,2,\ldots,2K$ may be easily constructed from the solution to the pure network problem, referred to as problem P4;

$$\{\min cx \mid Ax = b, \gamma \leq x \leq \theta\},$$

where the lower and upper bound n-vectors γ and θ are defined by

$$\begin{aligned} \gamma_k &= \theta_k = y_k, && k = 1,2,\ldots,K \\ \gamma_{K+k} &= \theta_{K+k} = y_k, && k = 1,2,\ldots,K \\ \gamma_k &= 0, \theta_k = u_k, && k = 2K+1,\ldots,n. \end{aligned}$$

Let Π be the vector of optimal dual variables associated with the conservation of flow constraints, $Ax = b$ in P4 and the arc associated with the variable x_j be incident from node j_f and incident to node j_t. The optimal dual variables for P3 are given by,

$$v_k = -\Pi_{j_f} + \Pi_{j_t} + c_k, \quad k = 1,2,\ldots,2K.$$

In using the subgradient optimization algorithm for the decomposed problem at each point y, the subgradient η can be calculated directly using the above development.

It is possible that moving a step along the negative subgradient yields a point which does not belong to the set S. As pointed out in Section II, this point is projected onto the set S by means of a projection operation in the algorithm. For this model, the projection operation decomposes on k so that $P[y] = (P[y_1], P[y_2], \ldots, P[y_K])$, where the projections $P[y_k]$ are defined by:

$$\begin{aligned} &\text{If } y_k < 0, && P[y_k] = 0. \\ &\text{If } y_k > u_k, && P[y_k] = u_k. \\ &\text{If } 0 \leq y_k \leq u_k, && P[y_k] = y_k. \end{aligned}$$

The subgradient optimization algorithm for problem P2 makes use of a lower bound, LBND, on the optimal objective value which is used in step size determination using a variant of scheme (iii) given in Section II, as well as in the termination criteria. Again, we assume that both bounds are greater than zero.

ALGORITHM 3: UPPER BOUND ALGORITHM

1 Initialization

Select $y \in S$ and construct γ and θ.

Initialize LBND, ε, q, n^*, $J \leftarrow 0$.

2 Find subgradient and step size

$J \leftarrow J + 1$.

Let $\bar{x}$ and Π be the vectors of optimal primal and dual variables

for Min $\{cx \mid Ax = b, \gamma \leq x \leq \theta\}$.

If $c\bar{x} >$ UBND,

$n' \leftarrow n' + 1$,

if $n' = n^*$, $q \leftarrow q/2$, $n' \leftarrow 0$;

otherwise,

$n' \leftarrow 0$

UBND $\leftarrow c\bar{x}$.

If (UBND - LBND) $\leq \varepsilon$(UBND) and $\bar{x}$ feasible, terminate with $\bar{x}$ optimal;

otherwise,

$v_k \leftarrow -\Pi_{k_f} + \Pi_{k_t} + c_k$, $k = 1,2,\ldots,2K$.

$\eta \leftarrow (v_1 + v_{K+1}, \ldots, v_K + v_{2K})$.

3 Move to new point

(a) $y \leftarrow P[y - q((UBND - LBND)/(\|\eta\|^2))\eta]$.

(b) If max $(q((UBND - LBND)/(\|\eta\|^2))\eta])_i < .01$ then terminate.

(c) Go to 2.

The use of the algorithm parameters q and n^* is to help condition the step sizes based on the relative norm of the subgradient with respect to the difference in the lower and upper bounds. The norm of the subgradient is dependent on the problem rather than the algorithm. That is, it is quite possible that the norm remains high throughout the algorithm. The initial choice of q is directed by an estimate of the maximum of the absolute values of the elements of the vector **d'** as well as the objective function coefficient associated with artificial variables in the solution of the pure network problem. When allocations yield infeasible solutions, the elements of η are large rendering $\|\eta\|^2$ very large. An initial value of q, if chosen arbitrarily, can be small, thus requiring more iterations since the improvement at each iteration is small. On the other hand, if the initial value of q is large, then for several sets of n^* iterations no improvement in the

objective occurs until the value of q becomes smaller. Here again, a secondary termination criterion in Step 3(b) is when further changes in the allocation in step 3(a) are minimal.

The modification required for the integer problem occurs only when the termination criteria have been met for the linear problem. The alternate formulation for the integer problem is obtained by requiring that the equal flow allocation, y be integral

$$\text{Minimize } g(y)$$
$$\text{s.t.} \quad y \in S^I$$

where for any vector $y = (y_1, y_2, \ldots, y_K)$,

$$g(y) = \{\min cx | Ax = b; 0 \le x \le u; x_k = x_{K+k} = y_k, k = 1,2,..,K\},$$

and,

$$S^I = \{y | 0 \le y_k \le u_k, \text{ for } k = 1,2,...,K \text{ and } y \text{ integer}\}.$$

Once termination occurs for the linear problem, the upper bound algorithm can be used by requiring that the projection in step 3 of the algorithm yield an integer equal flow allocation. Since the objective retains the piece-wise convex nature of the objective to the linear problem, the linear optimum obtained can be expected to be close to the integer optimum. Adjacent integer allocations can be expected to provide bounds on the integer optimum or else be near-feasible points for the integer problem.

5 The Algorithm

The solution of the equal flow problem using decomposition, as given in the previous section can be implemented without the lower bound procedure. It is also possible to implement the lower bound algorithm independently for the purpose of obtaining a lower bound on the optimal value of the equal flow problem. For the upper bound problem, some measure of the lower bound on the problem must be used to aid in termination. By merging the two procedures, an algorithm which adjusts the lower and upper bounds progressively can be used to advantage and tied to the accuracy desired for the solution. Not only can such a procedure be used for obtaining feasible solutions with relative ease, but it can also provide a measure of how close this solution is to the optimal.

The algorithm for the solution of the equal flow problem iterates between the lower bound procedure and the upper bound procedure. The lower and upper bounds, LBND and UBND, progressively become tighter, closing in on the optimal solution to the

problem. Each time the lower bound procedure is invoked, a maximum of ITERL iterations are performed. Each time the upper bound procedure is invoked, a maximum of ITERU iterations are performed. However, the initial invocations of the lower and upper bound algorithms are allowed to terminate using criteria in those algorithms as opposed to these iteration counts. The initial invocation of the Lagrangean dual is important primarily because it affords a very tight lower bound on the objective value of the integer or linear optimum and further it provides near-optimal values of the Lagrange multipliers. The near-optimal values of the Lagrange multipliers tend to aid the subgradient optimization of the decomposition model. The tuning parameters for the algorithm are as follows: ITERL, ITERU, m^*, n^*, and ε (the termination criterion.)

ALGORITHM 4: RELAXATION/DECOMPOSITION ALGORITHM FOR THE EQUAL FLOW PROBLEM

0 Initialization

Initialize ITERL, ITERU, ε.

$T \leftarrow 0$, $R \leftarrow 0$, $w \leftarrow 0$, UBND $\leftarrow \infty$, LBND $\leftarrow -\infty$.

Call ALGORITHM 2 and $y_k \leftarrow \min[\, u_k, (x'_k + x'_{K-k})/2 \,]$, $k = 1,2,\ldots,K$.

Call ALGORITHM 3

1 Compute Lower Bounds

(a) Call ALGORITHM 2 (Steps 2 and 3 (a)).

(b) $T \leftarrow T+1$

If $T <$ ITERL, then go to step 1 (a).

2 Compute Upper Bounds

(a) Call ALGORITHM 3 (Steps 2 and 3 (a)).

(b) $R \leftarrow R+1$

If $R <$ ITERU, then go to step 2 (a).

3 Reset iteration counts

$T \leftarrow 0$, $R \leftarrow 0$, and go to step 1.

6 Computational Experience

The computer implementation of the algorithm is written in standard FORTRAN (called EQFLO) and makes use of MODFLO [1] to solve pure network subproblems. Based on NETFLO [8], MODFLO is a set of subroutines which allows parametric changes in costs, bounds and/or requirements for a network problem and subsequent

reoptimization. Computational testing was carried out on the IBM 3081D at The University of Texas at Austin using the FORTVS compiler with OPT = 2. In order to assess the computational gains afforded by the decomposition/relaxation algorithm for the equal flow problem, each problem was solved using MPSX [7]. All MPSX solutions have been obtained on the IBM3081D at Southern Methodist University.

The algorithm has been tested on a set of 10 test problems generated by using NETGEN [9], and referred to by their NETGEN numbers. Of the 10 problems used, the first three are transportation problems (problems 5, 9, and 10), the next four are capacitated transshipment problems (problems 20, 21, 24, and 25) and the last three are uncapacitated transshipment problems (problems 28, 30, and 35). The test problems have between 200 and 1500 nodes, and between 1500 and 6600 arcs. For each problem, the first 2K arcs were paired to form K equal flow side constraints. In order to gauge the performance of the algorithm for various values of K, some of the problems were generated using the same base network problem data with K varying from 75 to 200.

The benchmark NETGEN problems have a specified percentage of arcs which are uncapacitated. For these arcs, the capacity was defined to be the maximum of all supplies and demands. For arcs in equal flow pairs which emanate from supply points, the capacity used is the supply at the point of incidence. Similarly, for arcs incident to demand points, the capacity used is the corresponding demand. If an equal flow pair is incident to a demand point or incident from a supply point, then the capacity assigned is the upper integer ceiling of half the corresponding requirement. Such allocation of capacity is prudent, allowing a tighter relaxation.

Table I details the computational testing of the algorithm with parameters and $m^* = 5$, $n^* = 10$, ITERU = ITERL = 10, $\varepsilon = .01$. For the test problems, EQFLO obtained feasible solutions whose objective function values were within 1% of the optimal in a fraction of the time required by MPSX to obtain an optimum. The table reports the total solution times required to produce an α percent solution for the linear equal flow problem and an integer solution. The number of lower and upper bound interations, respectively I and J, required are provided along with the norm, $\|d'\|$, obtained during subgradient optimization of the Lagrangean dual. Note that this norm typically provides a metric for gauging the difficulty of a specific problem instance. Because of the fact that the lower bound procedure does not enforce equal flow, the norm provides a measure of the infeasibililty of equal flow constraints, or the amount of imbalance which exists in the problem. The zero-tolerance used for flows on artificial arcs is .05. Of the 10 problems, feasible solutions were obtained for all linear problems.

Termination criteria employed for this computational testing are stringent and the decomposition algorithm continues to attempt improvement until not only the solution is within 1% of the lower bound, but also until changes in subsequent allocations are negligible (.001).

Initial allocations, as determined by $\mathbf{x}'$, obtain feasible solutions well within 1% of the lower bound obtained in 6 of the 10 problems. For the other problems, the initial allocation can be feasible or infeasible. Capacities in the randomly generated problems are such that infeasibilty occurs due to the following: When a particular level of allocation is enforced, the problem can become infeasible due to capacities falling below a level required to ensure all demand be met.

An upper limit of 5 iterations were allowed in performing integer equal flow allocations with the initial integer allocation obtained by truncating the optimal linear allocation. No more than 29 units of demand go unsatisfied corresponding to 99.99% feasible integer solutions well within 1% of the lower bound obtained. The trade-off between enforcing integer equal flows and 100% feasible solutions tips in favour of making use of near-feasible solutions, given the relative computational ease with which they are produced. Problem 5 was attempted with MPSX-MIP where integrality was only forced on the 75 equal flow pairs. *After over* 220 seconds the active branch-and-bound tree had over 2000 nodes and had not as yet obtained the first feasible integer solution. In less than 9 seconds, the decomposition procedure obtained an integer solution which satisfied 399,996 units of the 400,000 units of demand.

To determine the impact of an increase in the number of side constraints on problem characteristics and the algorithm, additional testing with 21, 24, and 28 is reported in Table II. Each of these base problems was used to generate equal flow problems with 75, 100, 150, and 200 equal flow constraints. As evident from the behavior of the norm of $\mathbf{d}'$, as the number of side constraints increases, more imbalance in the problem is introduced and in order to enforce equal flow, more effort is required. Problem 24 becomes infeasible once the number of side constraints enforced becomes 200. As would be expected, the algorithm expends more effort for the more tightly constrained problems, with the exception that it recognizes an infeasible problem readily. Again, for the problems which are feasible, near-feasible integer solutions are obtained in approximately 1%-60% of the time required to solve the linear problem using MPSX.

7 Summary and Conclusions

The equal flow problem lends itself to solution by decomposition and relaxation. The use of these techniques in the solution procedure developed is advantageous because the essential solution mechanism required is the solution of sequences of pure network problems. By dispensing with the working basis required by other techniques, not only are computational efficiencies afforded but the natural characteristics of the problem enhanced.

The algorithm has been shown to assist in solving the integer equal flow problem. The lower bound automatically produces integer flows and the projection of the sub-gradient in the upper bound routine is altered to require integrality on the equal flow allocation once a near-optimal linear solution has been obtained. The equal flow allocation for the linear model is expected to be close to the equal flow allocation for the integer model due to problem structure. Thus the solution procedure provides near-feasible, near-optimal solutions for the integer equal flow problem efficiently.

The structure of the equal flow problem provides a metric on the relative difficulty of any specific problem instance. The proposed solution procedure has the innate capability to distinguish between easy and difficult instances of an equal flow problem and thus can require only 1% of the MPSX time to solve an easy problem. As the number of equal flow constraints grows, a problem can become progressively infeasible, since the enforcement of equal flows can serve to reduce the capacity of a cut-set of the network to well below required levels for feasibilty. The development for the linear equal flow problem in this paper can be instructive in modelling and solving other network problems with specially structured side constraints such as proportional flow models used in manpower planning. The solution technique is best suited for a real-world situation in which one must quickly produce near-optimal, near-feasible solutions.

REFERENCES

[1] Ali, A., E. Allen, R. Barr, and J. Kennington, "Reoptimization Procedures for Bounded Variable Primal Simplex Network Algorithms," *European Journal of Operational Research*, 23, 256-263 (1986).

[2] Allen E., R. Helgason, J. Kennington, and B. Shetty, "A Generalization of Polyak's Convergence Result for Subgradient Optimization," Technical Report 85-OR-7, Department of Operations Research, Southern Methodist University, Dallas, Texas, 75275 (1985), to appear in *Mathematical Programming*.

[3] Barr, R., K. Farhangian, and J. Kennington, "Networks with Side Constraints: An LU Factorization Update," *The Annals of the Society of Logistics Engineers.*, 1, 1, 66-85 (1986).

[4] Beck, P., L. Lasdon, and M. Engquist, "A Reduced Gradient Algorithm for Nonlinear Network Problems," *ACM Transactions on Mathematical Software*, 9, 57-70 (1983).

[5] Carraresi, P. and G. Gallo, "Network Models for Vehicle and Crew Scheduling," *European Journal of Operational Research*," 16, 139-151 (1984).

[6] Glover, F., R. Glover, and F. Martinson, "The U. S. Bureau of Land Management's New NETFORM Vegetation Allocation System," Technical Report of the Division of Information Science Research, University of Colorado, Boulder, Colorado, 80309 (1982).

[7] IBM Mathematical Programming System Extended/370 Program Reference Manual, File No. S370-82, IBM Corp., White Plains, New York (1979).

[8] Kennington, J., and R. Helgason, *Algorithms for Network Programming*, John Wiley and Sons, New York (1980).

[9] Klingman, D., A Napier, and J. Stutz, "NETGEN: A Program for Generating Large Scale Minimum Cost Flow Network Problems," *Management Science*, 20, 814-821 (1974).

[10] Poljak, B. T., "A General Method of Solving Extremum Problems," *Soviet Mathematics Doklady*, 8, 3, 593-597 (1967).

[11] Shepardson, F., and R. Marsten, "A Lagrangean Relaxation Algorithm for the Two Duty Period Scheduling Problem," *Management Science*, 26, 274-281 (1980).

[12] Shetty, B., "The Equal Flow Problem," unpublished dissertation, Department of Operations Research, Southern Methodist University, Dallas, Texas, 75275 (1985).

[13] Shor, N., "On the Structure of Algorithms for the Numerical Solution of Optimal Planning and Design Problems," Dissertation, Cybernetics Institute, Academy of Sciences, U.S.S.R. (1964).

[14] Turnquist, M., and C. Malandraki, "Estimating Driver Costs for Transit Operations Planning," Joint National Meeting of ORSA/TIMS, Dallas (1984).

[15] Zangwill, W., *Nonlinear Programming: A Unified Approach*, Prentice Hall, Englewood Cliffs, New Jersey (1969).

Table I. Comparison of EQFLO with MPSX (All Problems Have 75 equal flow pairs).

NETGEN Number	Nodes	Arcs	MPSX Time	LINEAR α	‖d'‖	I	J	Time	INTEGER Infeas	Time
5	200	3100	11.4	0.15	377	151	1	8.6	4	8.8
9	300	6395	38.4	0.01	1296	145	1	19.6	3	19.9
10	300	6611	34.2	0.00	698	165	1	15.3	3	15.7
20	400	1484	34.8	0.10	4729	544	262	23.3	2	23.6
21	400	2904	73.8	0.00	1	6	1	00.7	1	1.1
24	400	1398	37.8	0.29	8875	280	498	21.4	3	21.7
25	400	2692	93.0	0.01	8356	148	1	5.7	1	6.3
28	1000	3000	52.8	0.14	3865	235	220	27.8	29	28.3
30	1000	4500	69.6	0.00	490	95	1	7.9	0	7.9
35	1500	5880	145.2	0.09	5386	218	151	46.7	6	47.7
			591.0					177.0		181.0

Times reported are in CPU seconds on an IBM3081D

Table II. Effect of Increasing the Number of Equal Flow Pairs.

NETGEN Number	Pairs	MPSX Time	LINEAR α	‖d'‖	I	J	Time	INTEGER Infeas	Time
21	75	73.8	0.00	1	6	1	00.7	1	1.1
21	100	64.8	0.00	1	6	1	00.8	1	1.1
21	150	83.4	0.08	880	130	202	14.6	2	15.1
21	200	76.2	0.47	2937	214	188	17.5	26	17.9
24	75	37.8	0.29	8875	280	498	21.4	3	21.7
24	100	36.0	0.53	18283	240	307	19.8	5	19.9
24	150	42.0	2.32	24867	258	505	30.7	37	30.9
24	200						infeasible		infeasible
28	75	52.8	0.14	3865	233	220	27.8	29	28.3
28	100	57.6	0.23	5206	213	182	28.3	35	28.9
28	150	65.4	0.30	6043	202	179	30.4	45	30.9
28	200	72.0	1.00	15206	224	423	50.5	49	51.0
		661.8					262.5		246.8

Times reported are in CPU seconds on an IBM3081D

CHAPTER 6

A PARALLELIZATION OF THE SIMPLEX METHOD

by

*R. V. Helgason *, J. L. Kennington *, and H. A. Zaki ***

February 1987.

ABSTRACT

This paper presents a parallelization of the simplex method for linear programming. Current implementations of the simplex method on sequential computers are based on a triangular factorization of the inverse of the current basis. An alternative decomposition designed for parallel computation, called the quadrant interlocking factorization, has previously been proposed for solving linear systems of equations. This research presents the theoretical justification and algorithms required to implement this new factorization in a simplex-based linear programming system.

* Department of Operation Research
Southern Methodist University
Dallas, TX 75275

** Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801

ACKNOWLEDGEMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR 83-0278, the Department of Defense under Contract Number MDA 903-86-C-0182, and Rome Air Development Center under Contract Number SCEEE PDP/86-75.

I. INTRODUCTION

The introduction of parallel computers into scientific computing in the past decade is the beginning of a new era. The invention of new algorithms will be required to ensure realization of the potential of these and future architectural improvements in computers. Already the use of parallel computers has given rise to studies in concurrency factors, vectorization, and asynchronous procedures. These have led to multifold increases in speed over conventional serial machines after the calculations have been rearranged to take advantage of the specific hardware. This paper presents a parallelization of the simplex algorithm for general linear programs. Our work begins with new results for solving systems of linear equations and is directed toward the hardware design currently adapted by Sequent Computer Systems, Inc. of Beaverton, Oregon.

The following notation is used throughout this paper. Let $B_{i:j,k:l}$ represent a submatrix of B composed of rows i through j and columns k through l. If $i=j$ $(k=l)$, we write $B_{i,k:l}$ $(B_{i:j,k})$. The j^{th} row (column) of B is denoted by $B_{j,*}$ $(B_{*,j})$. The i,j^{th} element of B is $B_{i,j}$.

The linear programming problem is represented mathematically as follows:

$$\text{minimize} \quad c^T x$$

$$\text{subject to} \quad Ax = b$$

$$0 \le x \le u,$$

where A is a known m by n matrix, all other quantities are conformable, and all vectors are known except x.

The upper bounded version of Dantzig's simplex method for solving the linear programming problem may be stated as follows:

Algorithm 1.1 : The Simplex Method

0. Initialization

Let $[x^B \mid x^N]$ be a basic feasible solution with $A = [B \mid N]$. Let the cost vector $[c^B \mid c^N]$ and bounds $[u^B \mid u^N]$ be partitioned similarly. Assume that B^{-1} is available in some factored form. Initialize *iter* to 0 and the reinversion frequency, *freq*.

1. Calculate the Dual Variables (BTRAN)

$$\pi \leftarrow c^B B^{-1}. \tag{1.1}$$

2. Pricing

Let $K_1 = \{j : x_j^N = 0 \text{ and } c_j^N - \pi N_{.j} < 0\}$,

and $K_2 = \{j : x_j^N = u_j^N \text{ and } c_j^N - \pi N_{.j} > 0\}$.

If $K_1 \cup K_2 = \Phi$, terminate with $[x^B \mid x^N]$ optimal;

otherwise, select $k \in K_1 \cup K_2$ and set

$$\delta \leftarrow \begin{cases} 1, \text{ if } k \in K_1 \\ -1, \textit{otherwise.} \end{cases}$$

3. Column Update (FTRAN)

$$y \leftarrow B^{-1} N_{.k} \tag{1.2}$$

4. Ratio Test

$$\Delta_1 \leftarrow \min_{sign(y_j) = sign(\delta)} \left\{ \frac{x_j^B}{|y_j|}, \infty \right\}$$

$$\Delta_2 \leftarrow \min_{sign(y_j) = sign(-\delta)} \left\{ \frac{u_j^B - x_j^B}{|y_j|}, \infty \right\}$$

$$\Delta \leftarrow \min \left\{ \Delta_1, \Delta_2, u_k^N \right\}.$$

5. Right Hand Side Update

$$x_k^N \leftarrow x_k^N + \Delta\delta$$

$$x^B \leftarrow x^B - \Delta\delta y.$$

If $\Delta = u_k^N$, return to 1.

6. Basis Inverse Update

Let p denote the index of x^B which produced Δ and set

$$\eta \leftarrow \begin{cases} -y_i/y_p \text{ , if } i \neq p \\ 1/y_p \text{ ; } otherwise, \end{cases}$$

$$E \leftarrow I - e_p e_p^T + \eta e_p^T$$

$$\bar{B}^{-1} \leftarrow EB^{-1}. \tag{1.3}$$

7. Reinversion Check

$iter \leftarrow iter + 1.$

If $mod(iter, freq) = 0$, then refactor $\bar{B}^{-1}$.

Return to 1 using $\bar{B}^{-1}$ as B^{-1}, the current basis inverse.

Two of the most common factorizations of the basis matrix inverse are the product form and the elimination form, which correspond to the methods for solution of linear equations known as Gauss-Jordan reduction and Gauss reduction (LU factorization), respectively, where L is a lower triangular matrix and U is an upper triangular matrix. The elimination form produces a sparser representation of the basis inverse than the product form, and accordingly leads to faster implementation of a simplex iteration and a considerable savings in storage.

Historically, the elimination form of the inverse, due to Markowitz [1957-1], was the first LU factorization method and was introduced to preserve sparsity during reinversion. However, once reinversion was completed further pivot operations were handled

using product form. Bartels and Golub proposed updating L and U in a numerically stable way, (see Bartels [1971-1]). Their updating scheme tends to promote the growth of nonzeros in U, leading to a potentially severe loss of sparsity. Forrest and Tomlin [1972-1] designed a different updating scheme for the triangular factors to preserve sparsity at some sacrifice in numerical stability. Subsequent implementation of the Bartels-Golub method, designed by Reid [1982-1] and Saunders [1976-1], combine the virtues of accuracy and speed.

Several parallel versions of the LU factorization algorithm for solving general linear systems of equations are presently available (Chen et al. [1984-1] and Dongarra and Sorensen [1984-2]). All versions are based on restructuring the original serial algorithm to reveal possible independent tasks that can be carried out concurrently.

Evans and Hatzopoulos [1979-1] proposed a matrix factorization, called the Quadrant Interlocking Factorization (QIF), as an appropriate tool for solving linear systems on parallel computers. The QIF is similar to the LU factorization, but is claimed to be more suitable for concurrent computation.

This paper presents a parallelization of the simplex method using the QIF. The outline of the paper is as follows. In Section II, the QIF is developed. An algorithm for updating the QIF of B^{-1} is presented in Section III. Mathematically, the problem is to efficiently obtain a factorization of $\bar{B}^{-1}$ (see step 6 of Algorithm 1.1) from the factorization of B^{-1}. In Section IV, we develop a parallelization of the reinversion routine used in step 7 and propose a parallel implementation of both the BTRAN and FTRAN operations of steps 1 and 3.

The parallel algorithms presented in this study are designed for a MIMD parallel computer that incorporates p identical processors sharing a common memory and capable of applying all their power to a single job in a timely and coordinated manner. The Balance Systems 8000 and 21000 from Sequent Computer Systems are examples of such machines.

II. THE QUADRANT INTERLOCKING FACTORIZATION

In this section we describe a matrix factorization suggested by Evans and Hatzopoulos [1979-1] known as *the Quadrant Interlocking Factorization (QIF)*. This decomposition is designed to solve linear systems on parallel computers (see Evans and Hatzopoulos [1979-1], Evans and Hadjidimos [1980-1], Evans [1982-1] and Feilmeier [1982-1]). The factors and some of their characteristics are described in Section 2.1. We show that any nonsingular matrix can be factorized into its QIF in two ways, the *Forward QIF* and the *Backward QIF* The factorization algorithms are developed in Sections 2.2 and 2.3. The relationship of quadrant and triangular matrices is presented in Section 2.4.

2.1 The Quadrant Interlocking Factors

Consider the following matrix

$$W = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ w_{2,1} & 1 & \cdots & 0 & w_{2,m} \\ w_{3,1} & w_{3,2} & \cdots & w_{3,m-1} & w_{3,m} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ w_{m-2,1} & w_{m-2,2} & \cdots & w_{m-2,m-1} & w_{m-2,m} \\ w_{m-1,1} & 0 & \cdots & 1 & w_{m-1,m} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{2.1}$$

Note that the non-arbitrary entries of W are given by

$$w_{i,j} = \begin{cases} 1, & i=j; \\ 0, & i=1,\ldots,[m/2],\ j=(i+1),\ldots,(m-i+1); \\ 0, & i=\bar{m},\ldots,m,\ j=m-i+1,\ldots,i-1; \end{cases} \tag{2.2}$$

where

$[x]$ = the largest integer not greater than the value of x

$\bar{m} = m + 1 - [m/2]$.

Also, consider the matrix

$$Z = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,m-1} & z_{1,m} \\ 0 & z_{2,2} & \cdots & z_{2,m-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & z_{m-1,2} & \cdots & z_{m-1,m-1} & 0 \\ z_{m,1} & z_{m,2} & \cdots & z_{m,m-1} & z_{m,m} \end{bmatrix}. \tag{2.3}$$

Note that

$$z_{i,j} = 0, \begin{cases} j = 1,\ldots,[(m-1)/2]\ ,\ i = j+1,\ldots,m-j; \\ j = [m/2]+2,\ldots,m\ ,\ i = m+2-j,\ldots,j-1. \end{cases} \tag{2.4}$$

Any square matrix may be partitioned by its diagonal and secondary diagonal into four quadrants. The potentially nonzero elements of W are in the left and right quadrants while those of Z are in the upper and lower quadrants. Therefore, we call any square matrix whose nonzero structure follows (2.1) and (2.2), or one that can be brought to such a form by row and/or column interchanges a *left-right quadrant (LRQ)* matrix. Similarly, any square matrix whose nonzero structure follows (2.3) and (2.4), or one that can be brought to such a form by row and/or column interchanges is called an *upper-lower quadrant (ULQ)* matrix . Examples of W and Z matrices for an odd and an even m are given below:

Example 2.1 (m=5)

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ w_{2,1} & 1 & 0 & 0 & w_{2,5} \\ w_{3,1} & w_{3,2} & 1 & w_{3,4} & w_{3,5} \\ w_{4,1} & 0 & 0 & 1 & w_{4,5} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} , Z = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} \\ 0 & z_{2,2} & z_{2,3} & z_{2,4} & 0 \\ 0 & 0 & z_{3,3} & 0 & 0 \\ 0 & z_{4,2} & z_{4,3} & z_{4,4} & 0 \\ z_{5,1} & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} \end{bmatrix} .$$

Example 2.2 (m=6)

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ w_{2,1} & 1 & 0 & 0 & 0 & w_{2,6} \\ w_{3,1} & w_{3,2} & 1 & 0 & w_{3,5} & w_{3,6} \\ w_{4,1} & w_{4,2} & 0 & 1 & w_{4,5} & w_{4,6} \\ w_{5,1} & 0 & 0 & 0 & 1 & w_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} , Z = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\ 0 & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\ 0 & 0 & z_{3,3} & z_{3,4} & 0 & 0 \\ 0 & 0 & z_{4,3} & z_{4,4} & 0 & 0 \\ 0 & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} & 0 \\ z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6} \end{bmatrix} .$$

Without loss of generality we assume that m is even. For linear programming, we can always append a nonbinding constraint so that the total number of constraints is even.

The set of all LRQ matrices of order m is denoted by $\{M_m^w\}$ and the set of all ULQ matrices of order m is denoted by $\{M_m^z\}$. Let A $\varepsilon R^{m,m}$ and $\bar{A}=A_{i,j}.e_i.e_j^T$. If $(\bar{A}+I)\varepsilon\{M_m^w\}$ we say that $A_{i,j}$ is a *W-element* ; otherwise, it is a *non-W-element* . Similarly, if $\bar{A}\,\varepsilon\{M_m^z\}$ we say that $A_{i,j}$ is a *Z-element* ; otherwise, it is a *non-Z-element.*

Proposition 2.1

$\{M_m^w\}$ and $\{M_m^z\}$ are closed under addition, scalar multiplication, multiplication and inversion .

(The proof of this Proposition may be found in Zaki [1986-1]).

2.2 The Forward Quadrant Interlocking Factorization Algorithm

In this section we present an algorithm which obtains the WZ factorization of any nonsingular matrix. That is, given a nonsingular matrix B, find W and Z such that $B = WZQ$, where Q is a permutation matrix. This factorization is analogous to the LU

factorization in common use in many production linear programming packages.

Definition 2.1

An elementary left-right quadrant (ELRQ) matrix of order m and index k is a matrix of the form:

$$N^k = I - u^k \cdot e_k^T - v^k \cdot e_l^T \tag{2.5}$$

where

$$l = m - k + 1 \qquad , k \in 1,2,...,(m/2)-1, \tag{2.6}$$

$$e_i^T \cdot u^k = 0 \quad \text{and} \quad e_i^T \cdot v^k = 0 \quad \text{for } i = 1,2,...,k,l,l+1,...,m. \tag{2.7}$$

The conditions (2.7) require that the first k and last k components of u^k and v^k be zero, that is, u^k and v^k have the form:

$$u^k = (0,0,...,0,u_{k+1}^k, u_{k+2}^k, \ldots, u_{m-k}^k, 0,0,...,0)^T \tag{2.8}$$

$$v^k = (0,0,...,0,v_{k+1}^k, v_{k+2}^k, \ldots, v_{m-k}^k, 0,0,...,0)^T. \tag{2.9}$$

In general an ELRQ matrix of order m and index k has the form depicted in Figure 2.1. Thus, an ELRQ matrix of index k is a LRQ matrix whose only nonidentity columns are columns k and l ($l=m-k+1$). ELRQ matrices are easily inverted. It is apparent that

$$\left[N^k\right]^{-1} = I + u^k \cdot e_k^T + v^k \cdot e_l^T \tag{2.10}$$

which is also an ELRQ matrix of index k.

Proposition 2.2

Let

$$N^{(k)} = N^1 N^2 \cdots N^k \tag{2.11}$$

where N^i is an ELRQ matrix of index i , i=1,2,...,k. Then $N^{(k)}$ is a LRQ matrix whose j^{th} and $(m-j+1)^{st}$ columns are those of N^j.

(The proof of this Proposition may be found in Zaki [1986-1]).

Definition 2.2

A partially reduced upper-lower quadrant (PRULQ) matrix of index k and order m is a square matrix whose non-Z elements are zero in columns 1 through $k-1$ and $l+1$ through m, where $k = 1,2,...,m/2$ and $l = m-k+1$. Its general form is shown in Figure 2.2. Note that B^1 has no special zero structure and $B^{m/2}$ is an ULQ matrix.

Proposition 2.3

Let B^k be a PRULQ matrix of index k. If B^k is nonsingular then there exist j_1 and j_2 such that $k \le j_1 < j_2 \le l$ and

$$\delta = B^k_{k,j_1} \cdot B^k_{l,j_2} - B^k_{k,j_2} \cdot B^k_{l,j_1} \neq 0. \tag{2.12}$$

Proof

Suppose $\delta=0$ for every $k \le j_1 < j_2 \le l$. Then $B^k_{k,.}$ must be a multiple of $B^k_{l,.}$. This contradicts the assumption that B^k is nonsingular.

Permuting the columns of a PRULQ matrix so that certain elements provide a nonsingular 2x2 submatrix is analogous to interchanging rows and columns in matrix inversion to obtain a nonzero pivot element. Now, let B^k be a nonsingular PRULQ matrix of index k. Let j_1 and j_2 satisfy Proposition 2.3 and define Q^k to be the permutation matrix such that

$$\bar{B}^k = B^k \, Q^k$$

where

$$\bar{B}^k_{.,k} = B^k_{.,j_1} \quad \text{and} \quad \bar{B}^k_{.,l} = B^k_{.,j_2} . \tag{2.13}$$

Let A be any square matrix of order m and let $k \in \{1,...,m/2\}$. Define $S^k(A)$ to be the following 2x2 principal submatrix of A

$$S^k(A) = \begin{bmatrix} A_{k,k} & A_{k,l} \\ A_{l,k} & A_{l,l} \end{bmatrix} \tag{2.14}$$

where $l = m-k+1$. Using these definitions and Proposition 2.3, it is clear that $\bar{B}^k = B^k \, Q^k$ is a nonsingular PRULQ matrix of index k and $S^k(\bar{B}^k)$ is nonsingular.

We now show how one may transform a PRULQ matrix of index k into a PRULQ

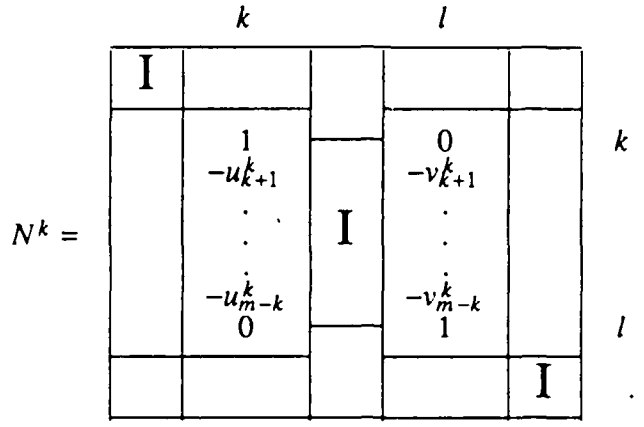


Figure 2.1. Illustration of the ELRQ matrix of order m and index k.

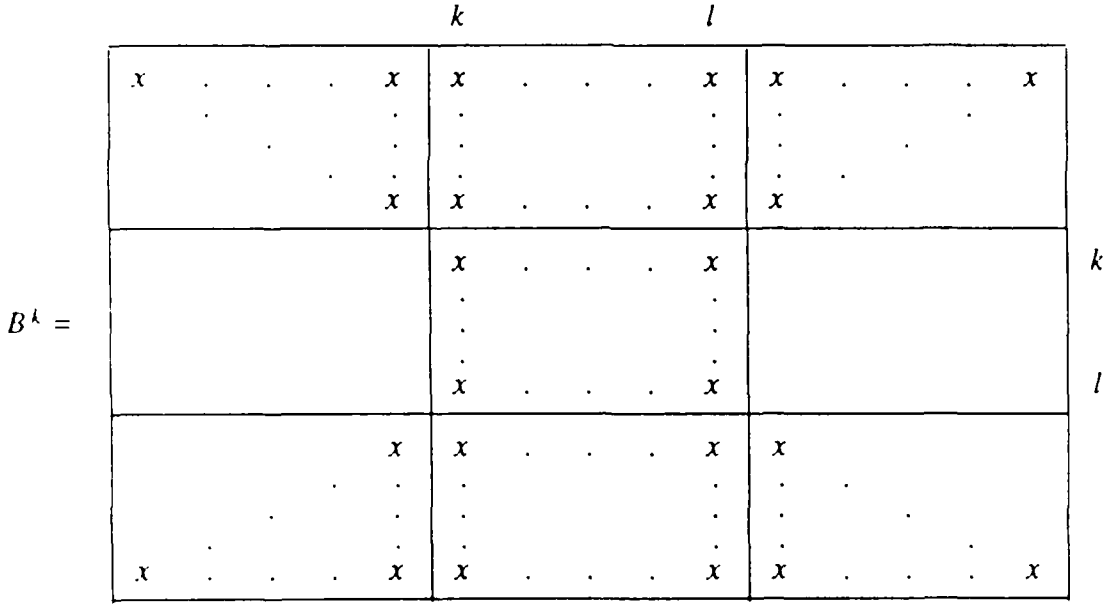


Figure 2.2. Illustration of a PRULQ matrix of order m and index k.

matrix of index $k+1$.

Proposition 2.4

Let B^k be a nonsingular PRULQ matrix of index k and let Q^k be the permutation matrix that interchanges columns k and $m-k+1$ with columns j_1 and j_2, respectively, where j_1 and j_2 are obtained so that they satisfy Proposition 2.3. Let N^k be an ELRQ matrix of index k whose u^k and v^k vectors are determined by solving the following $(m-2k)$ 2x2 linear systems

$$\begin{bmatrix} u_i^k & v_i^k \end{bmatrix} \quad S^k(\bar{B}^k) = \begin{bmatrix} \bar{B}_{i,k}^k & \bar{B}_{i,l}^k \end{bmatrix} \qquad , i=k+1,\ldots,m-k. \tag{2.15}$$

Then $B^{k+1} = N^k B^k Q^k$ is a nonsingular PRULQ matrix of index $k+1$.

Proof

Since $\bar{B}^k$ is nonsingular and N^k is nonsingular, then B^{k+1} is nonsingular. B^{k+1} is a PRULQ matrix of index $k+1$ if all non-Z-elements in columns 1 through k and l through m are zero. Since $\bar{B}^k$ is a PRULQ matrix of index k , we only need to show that the effect of N^k on $\bar{B}^k$ is to zero out the non-Z-elements in columns k, l. To show this, we begin by rewriting (2.15) as

$$\begin{bmatrix} u_i^k & v_i^k \end{bmatrix} \begin{bmatrix} \bar{B}_{k,k}^k & \bar{B}_{l,k}^k \\ \bar{B}_{k,l}^k & \bar{B}_{l,l}^k \end{bmatrix} = \begin{bmatrix} \bar{B}_{i,k}^k & \bar{B}_{i,l}^k \end{bmatrix}$$

or for $i = k+1,k+2,\ldots,m-k$

$$u_i^k \cdot \bar{B}_{k,k}^k + v_i^k \cdot \bar{B}_{l,k}^k = \bar{B}_{i,k}^k \tag{2.16}$$

$$u_i^k \cdot \bar{B}_{k,l}^k + v_i^k \cdot \bar{B}_{l,l}^k = \bar{B}_{i,l}^k. \tag{2.17}$$

We now consider the non-Z-elements of $B_{.,k}^{k+1}$.

For $i = k+1,k+2,\ldots,m-k$

$$\begin{aligned} B_{i,k}^{k+1} &= N_{i,.}^k \cdot \bar{B}_{.,k}^k \\ &= -u_i^k \cdot \bar{B}_{k,k}^k - v_i^k \cdot \bar{B}_{l,k}^k + \bar{B}_{i,k}^k = 0 \quad \textit{by (2.16).} \end{aligned} \tag{2.18}$$

$$\begin{aligned} B_{i,l}^{k+1} &= N_{i,.}^k \cdot \bar{B}_{.,l}^k \\ &= -u_i^k \cdot \bar{B}_{k,l}^k - v_i^k \cdot \bar{B}_{l,l}^k + \bar{B}_{i,l}^k = 0 \quad \textit{by (2.17).} \end{aligned} \tag{2.19}$$

$$B^{k+1}_{i,j} = N_{i,*} \cdot \bar{B}^{k}_{*,j} = 0 \quad \text{for } j=1,...,k-1 \text{ and } l+1,...,m. \tag{2.20}$$

Also we note that the desired zeros created in earlier stages in $\bar{B}^k$ are not affected by N^k, since for $i=1,...,k-1,l+1,...,m$

$$B^{k+1}_{i,*} = N^{k}_{i,*} \cdot \bar{B}^{k} = e_i \cdot \bar{B}^{k} = \bar{B}^{k}_{i,*}. \tag{2.21}$$

From (2.18) through (2.21) we conclude that B^{k+1} is a PRULQ matrix of index $k+1$.

Given the above definitions, the forward quadrant interlocking factorization algorithm may be stated as follows.

Algorithm 2.1 : The Forward Quadrant Interlocking Factorization

Let $B \in R^{m,m}$. The following steps decompose B to its quadrant interlocking factors with $B = W\ Z\ Q$.

Initialize

$B^1 = B$,
$K = m/2$.

Main Loop

For $k = 1,2,....,K-1$

1. Column Permutation

   Find j_1 and j_2 satisfying Proposition 2.3.

   If none exists, then terminate with the conclusion that B is singular.

   Otherwise, construct Q^k using j_1 and j_2.

2. Compute the vectors u^k, v^k

   by solving the $(m-2k)$ 2x2 linear systems, (2.15).

3. Construct N^k

$$N^k = I - u^k \cdot e_k^T - v^k \cdot e_l^T.$$

4. Construct B^{k+1}

$$B^{k+1} = N^k\ B^k\ Q^k.$$

Next k

Proposition 2.5

Let B be a nonsingular matrix of size m. Then *Algorithm 2.1* decomposes B to its forward quadrant interlocking factors ,

$$B = W\ Z\ Q \tag{2.22}$$

where

(1) $W \in \{M_m^w\}$, $W = (N^{K-1}N^{K-2}\cdots N^1)^{-1}$,
(2) $Z \in \{M_m^z\}$, $Z = B^K$,and
(3) Q is a permutation matrix , $Q = (Q^1Q^2\cdots Q^{K-1})^{-1}$.

Proof

Let $B^1 = B$. Applying Proposition 2.4 for k =1,2,...,$(m/2)-1$, we obtain

$$B^K = N^{K-1}\,N^{K-2}\cdots N^1\,B^1\,Q^1\cdots Q^{K-2}\,Q^{K-1}, \tag{2.23}$$

where B^K is an ULQ matrix, N^j, j = 1,...,$K-1$ are ELRQ matrices as computed in (2.15) and Q^j are permutation matrices. From (2.23),

$$B^1 = (N^{K-1}\,N^{K-2}\cdots N^1)^{-1}\,B^K\,(Q^1\cdots Q^{K-2}\,Q^{K-1})^{-1}. \tag{2.24}$$

Let $N^{(K-1)} = (N^{K-1}\,N^{K-2}\cdots N^1)^{-1}$. By Proposition 2.2 $N^{(K-1)}$ is a LRQ matrix. Also, let $Q^{(K-1)} = (Q^1\cdots Q^{K-2}\,Q^{K-1})^{-1}$. Since the product of permutation matrices is a permutation matrix, $Q^{(K-1)}$ is a permutation matrix. Thus, (2.24) can be written as

$$B^1 = B = N^{(K-1)}\ B^K\ \ Q^{(K-1)}, \tag{2.25}$$

and (2.22) follows by setting $W = N^{(K-1)}$, $Z = B^K$, and $Q = Q^{(K-1)}$ in (2.25).

Proposition 2.6

Algorithm 2.1 without column permutations requires

$$m^3/3 + m^2/2 - 4m/3$$

multiplications on a sequential machine.

Proof

Ignoring column permutations, we trace the operations in the main loop excluding step 1

The number of multiplications to compute u^k and v^k

$$= \sum_{k=1}^{K-1} [\, 2 + 6(m-2k) \,]$$
$$= m + 3m(m-2)/2. \qquad (2.26)$$

The number of multiplications to compute B^{k+1}

$$= 2. \sum_{k=1}^{K-1} (m-2k)^2$$
$$= m.(m-1).(m-2)/3. \qquad (2.27)$$

Summing (2.26) and (2.27) we obtain the specified total number of multiplications.

In Algorithm 2.1 the columns of the PRULQ matrix are permuted to find a 2x2 matrix with a nonzero determinant. There are obvious alternatives that may be used. To ensure numerical stability for instance, we may find the matrix whose determinant has the largest absolute value, or the matrix that has the smallest condition number. Another approach is to permute the rows of the PRULQ matrix to find the required nonsingular 2x2 matrix attempting to minimize fill-in in the nonpivot rows. Both row and/or column permutations can be selected on numerical stability and/or sparsity grounds.

2.3 The Backward Quadrant Interlocking Factorization Algorithm

Unlike the triangular factors (L,U) of a matrix, the quadrant interlocking factors (W,Z) possess different potential density. That is, the number of potentially nonzero elements in W is different than that in Z. In this section we present an algorithm which obtains the ZW factorization of any nonsingular matrix. We refer to this algorithm as the *Backward* QIF algorithm, as opposed to the *Forward* QIF algorithm of Section 2.2 that produces the WZ factorization. The development of this algorithm is very similar to the previous one. The proofs of Propositions 2.7 through 2.10 in this section, use arguments similar to those used in Propositions 2.2 through 2.5 and hence are omitted.

Definition 2.3

An elementary upper lower quadrant (EULQ) matrix of order m and index k is a matrix of the form :

$$M^k = I_m - r^k \cdot e_k^T - s^k \cdot e_l^T - e_k \cdot e_k^T - e_l \cdot e_l^T \tag{2.28}$$

where

$$l = m - k + 1, k \in 1,2, \cdots, m/2,$$
$$e_i^T \cdot r^k = 0 \quad \text{and} \quad e_i^T \cdot s^k = 0 \quad \text{for } i = k+1, k+2, \ldots, l. \tag{2.29}$$

The conditions (2.29) require that components $k+1$ through $m-k$ of r^k and s^k be zero, which are the non-Z-elements of r^k and s^k in M^k. That is, r^k and s^k have the form :

$$r^k = (r_1^k, \ldots, r_k^k, 0, \ldots, 0, r_l^k, \ldots, r_m^k)^T \tag{2.30}$$
$$s^k = (s_1^k, \ldots, s_k^k, 0, \ldots, 0, s_l^k, \ldots, s_m^k)^T. \tag{2.31}$$

Thus, an EULQ matrix of index k and order m is an ULQ matrix whose only nonidentity columns are columns k and l ($l=m-k+1$). In general, it has the form depicted in Figure 2.3.

The set of all nonsingular EULQ matrices is closed under inversion, and the inverse of any nonsingular EULQ matrix of index k is another EULQ matrix of index k.

Proposition 2.7

Let $M^{(k)} = M^k M^{k-1} \cdots M^1$ where M^i is an EULQ matrix of index i , i=1,2,....k. Then $M^{(k)}$ is a ULQ matrix whose j^{th} and $(m-j+1)^{st}$ columns are those of M^j , $j=1,2,\ldots,m/2$. The proof is similar to that of Proposition 2.2.

Definition 2.4

A partially reduced left-right quadrant (PRLRQ) matrix of index k and order m is a square matrix whose non-W-elements are zero in columns $k+1$ through $m-k$. Note that $B^{m/2}$ has no special zero structure and B^1 is an LRQ matrix. In general, a PRLRQ matrix is of the form shown in Figure 2.4.

$$M^k = \begin{array}{c|c|c|c|c|c} & k & & l & & \\ \hline & -r_1^k & & -s_1^k & & \\ \mathrm{I} & \vdots & & \vdots & & \\ & -r_k^k & & -s_k^k & & k \\ \hline & 0 & & 0 & & \\ & \vdots & \mathrm{I} & \vdots & & \\ & 0 & & 0 & & \\ \hline & -r_l^k & & -s_l^k & & \\ & \vdots & & \vdots & \mathrm{I} & \\ & -r_m^k & & -s_m^k & & l \end{array}$$

Figure 2.3. Illustration of the EULQ matrix of order m and index k.

$$B^k = \begin{array}{c|c|c|c} & k & & l \\ \hline x \ \cdots \ x & & & x \ \cdots \ x \\ \vdots \quad \vdots & & & \vdots \quad \vdots \\ x \ \cdots \ x & & & x \ \cdots \ x \quad k \\ \hline x \ \cdots \ x & \begin{matrix} 1 & & & & & & & 0 \\ x & \cdot & & & & & & x \\ \cdot & \cdot & \cdot & & & & \cdot & \cdot \\ \cdot & & x & 1 & 0 & x & & \cdot \\ \cdot & & x & 0 & 1 & x & & \cdot \\ & \cdot & \cdot & & & \cdot & \cdot & \\ x & \cdot & & & & & x & \\ 0 & & & & & & 1 & \end{matrix} & x \ \cdots \ x \\ \hline x \ \cdots \ x & & & x \ \cdots \ x \quad l \\ \vdots \quad \vdots & & & \vdots \quad \vdots \\ x \ \cdots \ x & & & x \ \cdots \ x \end{array}$$

Figure 2.4. Illustration of a PRLRQ matrix of order m and index k.

Proposition 2.8

Let B^k be a PRLRQ matrix of index k. If B^k is nonsingular then there exist j_1 and j_2 such that $1 \le j_1 \le k$ and $l \le j_2 \le m$ and

$$\delta = B^k_{k,j_1} \cdot B^k_{l,j_2} - B^k_{k,j_2} \cdot B^k_{l,j_1} \neq 0 \tag{2.32}$$

The proof is similar to that of Proposition 2.3.

Now let j_1 and j_2 satisfy Proposition 2.8 and define P^k to be a permuted identity matrix with column j_1 in the k^{th} position and j_2 in the l^{th}. Let B^k be a nonsingular PRLRQ matrix of index k. Obviously, $\hat{B^k} = B^k P^k$ is a nonsingular PRLRQ matrix of index k , and $S^k(\hat{B^k})$ is nonsingular.

Using M^k of (2.28) and the P^k defined above, the elimination operation needed to reduce a PRLRQ matrix of index k a step further is given by the following Proposition.

Proposition 2.9

Let B^k be a nonsingular PRLRQ matrix of index k , let j_1 and j_2 satisfy Proposition 2.8, let P^k be the permutation matrix that permutes columns k and j_1 and columns $m-k+1$ and j_2. Let M^k be an EULQ matrix of index k whose r^k,s^k vectors are determined by solving the following $2k-2$ linear systems

$$\begin{bmatrix} r_i^k & s_i^k \end{bmatrix} . S(\hat{B^k}) = \begin{bmatrix} \hat{B}^k_{i,k} & \hat{B}^k_{i,l} \end{bmatrix} \quad , i=1,...,k-1 \text{ and } l+1,...,m \tag{2.33}$$

along with the system

$$\begin{bmatrix} r_k^k & s_k^k \\ r_l^k & s_l^k \end{bmatrix} = \left[S^k(\hat{B^k}) \right]^{-1}. \tag{2.34}$$

Then $B^{k-1} = M^k B^k P^k$ is a nonsingular PRLRQ matrix of index $k-1$.

Given the above definitions, we may state the backward QIF algorithm as follows:

Algorithm 2.2 : The Backward Quadrant Interlocking Factorization

Let $B \in R^{m,m}$. The following steps decompose B to its QIF with $B = Z\ W\ P$.

Initialize

$$B^{m/2} = B,$$
$$K = m/2.$$

Main Loop

For $k = K, K-1, K-2, \ldots, 1$

1. Column Permutation

   Find j_1 and j_2 satisfying Proposition 2.8.

   If none exists, then terminate with the conclusion that B is singular.

   Otherwise, construct P^k using j_1 and j_2.

2. Compute the vectors r^k , s^k

   by solving the $(2k-1)$ 2x2 linear systems (2.33) and (2.34).

3. Construct M^k

$$M^k = I_m - r^k \cdot e_k^T - s^k \cdot e_l^T - e_k \cdot e_k^T - e_l \cdot e_l^T.$$

4. Construct B^{k-1}

$$B^{k-1} = M^k \; B^k \; P^k.$$

Next k

Proposition 2.10

Let B be a nonsingular matrix of size m. Then *Algorithm 2.2* decomposes B to its backward QIF,

$$B = Z \; W \; P \tag{2.35}$$

where

(1) $Z \in \{M_m^z\}$, $Z = (M^1 M^2 \cdots M^K)^{-1}$,
(2) $W \in \{M_m^w\}$, $W = B^1$,and
(3) P is a permutation matrix , $P = (P^K \cdots P^1)^{-1}$.

The proof is similar to that of Proposition 2.5.

As with the Forward QIF Algorithm, row and/or column permutations can be adopted to ensure numerical stability and/or sparse factors.

2.4 Some Characteristics of Quadrant Matrices

In this section we reveal a relationship between the quadrant and triangular matrices, which has not previously appeared in the open literature (e.g. Evans and Hatzopoulos [1979-1], Evans and Hadjidimos [1980-1], Evans [1982-1], Feilmeier [1982-1], Hellier [1982-1], and Shanehchi and Evans [1982-1]). A permutation algorithm that restructures any quadrant matrix as a block triangular one is presented.

Consider the following matrices

$$\hat{Z} = \begin{bmatrix} & & & & \\ x & x & & & \\ x & x & & & \\ x & x & x & x & \\ x & x & x & x & \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ x & x & x & x & x\ x \\ x & x & x & x & x\ x \end{bmatrix} \quad , \quad \hat{W} = \begin{bmatrix} 1 & 0 & x & x & . & x & x \\ & 1 & x & x & . & x & x \\ & & 1 & 0 & . & x & x \\ & & & 1 & . & x & x \\ & & & & . & . & . \\ & & & & & . & . \\ & & & & & . & . \\ & & & & & 1 & 0 \\ & & & & & & 1 \end{bmatrix} . \qquad (2.36)$$

Where x stands for a potential nonzero element. Note that $\hat{Z}$ is a lower Hessenberg matrix with a special zero distribution on the superdiagonal. Also, $\hat{W}$ is a unit upper triangular matrix with special zero distribution on the superdiagonal.

Now we present an algorithm that relates W of (2.1) and Z of (2.3) to $\hat{W}$ and $\hat{Z}$ of (2.36).

Algorithm 2.3 : The Permutation Algorithm

Let R, S, and T be square matrices of order m, where R is the input matrix to the algorithm and T is the output matrix. The following algorithm permutes the columns and rows of R such that:

(a) if R is a LRQ matrix then T is a $\hat{W}$ of (2.36), and

(b) if R is a ULQ matrix then T is a $\hat{Z}$ of (2.36).

1. Column Permutation

For $j = 1,2,\ldots,m/2$

$$S_{.,m-2j+1} \leftarrow R_{.,j}$$
$$S_{.,m-2j+2} \leftarrow R_{.,m-j+1}$$

Next j

2. Row Permutation

For $i=1,2,...,m/2$

$$T_{m-2i+1,.} \leftarrow S_{i,.}$$
$$T_{m-2i+2,.} \leftarrow S_{m-i+1,.}$$

Next i

An example of the permutation algorithm is given below for $m=6$.

Example 2.3 ($m=6$)

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ w_{2,1} & 1 & 0 & 0 & 0 & w_{2,6} \\ w_{3,1} & w_{3,2} & 1 & 0 & w_{3,5} & w_{3,6} \\ w_{4,1} & w_{4,2} & 0 & 1 & w_{4,5} & w_{4,6} \\ w_{5,1} & 0 & 0 & 0 & 1 & w_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \hat{W} = \begin{bmatrix} 1 & 0 & w_{3,2} & w_{3,5} & w_{3,1} & w_{3,6} \\ 0 & 1 & w_{4,2} & w_{4,5} & w_{4,1} & w_{4,6} \\ 0 & 0 & 1 & 0 & w_{2,1} & w_{2,6} \\ 0 & 0 & 0 & 1 & w_{5,1} & w_{5,6} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$Z = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\ 0 & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\ 0 & 0 & z_{3,3} & z_{3,4} & 0 & 0 \\ 0 & 0 & z_{4,3} & z_{4,4} & 0 & 0 \\ 0 & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} & 0 \\ z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6} \end{bmatrix}, \hat{Z} = \begin{bmatrix} z_{3,3} & z_{3,4} & 0 & 0 & 0 & 0 \\ z_{4,3} & z_{4,4} & 0 & 0 & 0 & 0 \\ z_{2,3} & z_{2,4} & z_{2,2} & z_{2,5} & 0 & 0 \\ z_{5,3} & z_{5,4} & z_{5,2} & z_{5,5} & 0 & 0 \\ z_{1,3} & z_{1,4} & z_{1,2} & z_{1,5} & z_{1,1} & z_{1,6} \\ z_{6,3} & z_{6,4} & z_{6,2} & z_{6,5} & z_{6,1} & z_{6,6} \end{bmatrix}.$$

This clearly shows that the quadrant matrices are permuted block triangular matrices with blocks of size 2. That is, the Forward (Backward) Quadrant Interlocking factorization is equivalent to a block Doolittle (Crout) decomposition with blocks of size 2.

On sequential computers, a QIF is not expected to be faster than any triangular

decomposition. Since computing the entries of the factors by solving 2x2 systems requires more operations, as shown in Proposition 2.6. Also, finding a nonsingular 2x2 submatrix is more expensive than finding a nonzero element. However, on parallel computers, the QIF is expected to be competitive. Since the number of entries that can be produced concurrently in every stage is doubled, and the number of stages is halved as compared to a triangular factorization algorithm. Therefore, we may view the column permutation step in Algorithms 2.1 and 2.2 searching for a nonsingular 2x2 submatrix as a computation decoupling price we pay for the concurrency gained in steps 2-4.

Determining the relationship between quadrant and triangular matrices is a key observation that we will use in the following section to design appropriate updating scheme for the quadrant interlocking factors of the basis matrix in the simplex method.

III. UPDATING THE QIF OF THE BASIS

At the beginning of a simplex iteration, suppose the basis has the form

$$B = Z\ W\ R, \tag{3.1}$$

where we assume forms (2.36) for Z and W, and R is a permutation matrix. When the entering column $A_{.j}$ replaces the leaving column $B_{.p}$ at the end of the simplex iteration, we have a new basis matrix $\bar{B}$ which is related to the previous basis matrix B by the formula

$$\bar{B} = B\ E \tag{3.2}$$

where E is an eta matrix whose p^{th} column is $(B^{-1} A_{.j})$, and all other columns are the identity columns. From (3.1) and (3.2) $\bar{B}$ can be written as

$$\bar{B} = Z\ W\ R\ E. \tag{3.3}$$

An *updating scheme* is a sequence of operations applied to the right side of (3.3) to return it to the form given by (3.1), i.e.

$$\bar{B} = \bar{Z}\ \bar{W}\ \bar{R}, \tag{3.4}$$

where $\bar{W}$, $\bar{Z}$ are the new Q.I. factors and $\bar{R}$ is a permutation matrix. We present an algorithm designed to derive (3.4) given (3.3). It is similar to the Forrest-Tomlin [1972-1] update for the triangular factors of the basis. Since the spike is in W, our strategy is to reduce the spiked W, i.e., WE, to an LRQ matrix using elementary ULQ matrices. The following algorithm exploits the triangular form of W and the existence of 2x2 identity blocks on the diagonal of W.

In this presentation we use the term *brother columns (rows)* to indicate columns (rows) that have the same potential nonzero structure, execluding the diagonal entries in case of LRQ matrices. Thus, for LRQ matrices in the form of (2.36) columns (rows)

$i, i+1$ are brother columns (rows) for $i = 1,3, \cdots, m-1$.

The first step of this scheme is a column permutation followed by a row permutation. In Figure 3.1 an example is presented to illustrate this step, in which R of (3.3) permutes columns 2 and 4 of W and x stands for potentially nonzero elements. Thus, W and WR are as illustrated in Figure 3.1 (a) and (b). From (3.3) we obtain

$$\begin{aligned} Z^{-1}\bar{B} &= W\ R\ E \\ &= \hat{S}, \end{aligned}$$

where $\hat{S}$ is illustrated in Figure 3.1 (c) and y stands for the elements of the column vector $(Z^{-1}A_{\cdot,j})$. Note that if $(Z^{-1}A_{\cdot,j})$ has the same zero structure as $W_{\cdot,q}$, then the new factors are immediately available. That is, $\bar{W}$ is $\hat{S}$ and $\bar{Z}$ is Z. If this is not the case, we place $\hat{S}$ in a spiked-W form S as shown in Figure 3.1 (d), by applying the column permutation R^{-1} to $\hat{S}$ to undo the effect of R. That is,

$$\begin{aligned} Z^{-1}\bar{B}\,R^{-1} &= W\ R\ E\ R^{-1} \\ &= \hat{S}\ R^{-1} \\ &= S. \end{aligned} \tag{3.5}$$

Suppose $q < m-1$. We apply the column permutation $\hat{R}$ to S, placing the spike and the brother of the leaving column in the positions m and $m-1$, respectively, and moving all intervening columns forward to produce the matrix H^q, as illustrated in Figure 3.1(e). We then apply the row permutation $\hat{R}^{-1}$ to H^q placing the q^{th} row and its brother row in positions m and $m-1$, respectively, moving all intervening rows two places up to produce the matrix $H^{\bar{q}}$ as shown in Figure 3.1 (f), where

$$\bar{q} = \begin{cases} q, \text{ if } q \textit{ is odd}; \\ q-1, \text{ if } q \textit{ is even}. \end{cases}$$

Note that $\bar{q}$ is odd. Of course, if $q \geq m-1$, then $\hat{R} = I$. Now (3.5) becomes

$$\begin{aligned} \hat{R}^{-1}Z^{-1}\bar{B}\,R^{-1}\hat{R} &= \hat{R}^{-1}\,W\ R\ E\ R^{-1}\hat{R} \\ &= \hat{R}^{-1}\hat{S}\,R^{-1}\hat{R} \\ &= \hat{R}^{-1}S\,\hat{R} \end{aligned}$$

1oxxxxxx	1xxoxxxx	1yxoxxxx	1oxyxxxx	1oxxxxxy	1oxxxxxy
o1xxxxxx	oxx1xxxx	oyx1xxxx	o1xyxxxx	o1xxxxxy	o1xxxxxy
oo1oxxxx	oo1oxxxx	oy1oxxxx	oo1yxxxx	ooxxxx1y	oo1oxxoy
ooo1xxxx	o1ooxxxx	oyooxxxx	oooyxxxx	ooxxxxoy	ooo1xxoy
oooo1oxx	oooo1oxx	oyoo1oxx	oooy1oxx	oo1oxxoy	oooo1ooy
ooooo1xx	ooooo1xx	oyooo1xx	oooyo1xx	ooo1xxoy	ooooo1oy
oooooo1o	oooooo1o	oyoooo1o	oooyoo1o	oooo1ooy	ooxxxx1y
ooooooo1	ooooooo1	oyooooo1	oooyooo1	ooooo1oy	ooxxxxoy
W	WR	$\hat{S}$	S	H^q	$H^{\bar{q}}$
(a)	(b)	(c)	(d)	(e)	(f)

Figure 3.1. Illustration of the double column and row permutation ($m=8, p=2, q=4, \bar{q}=3$).

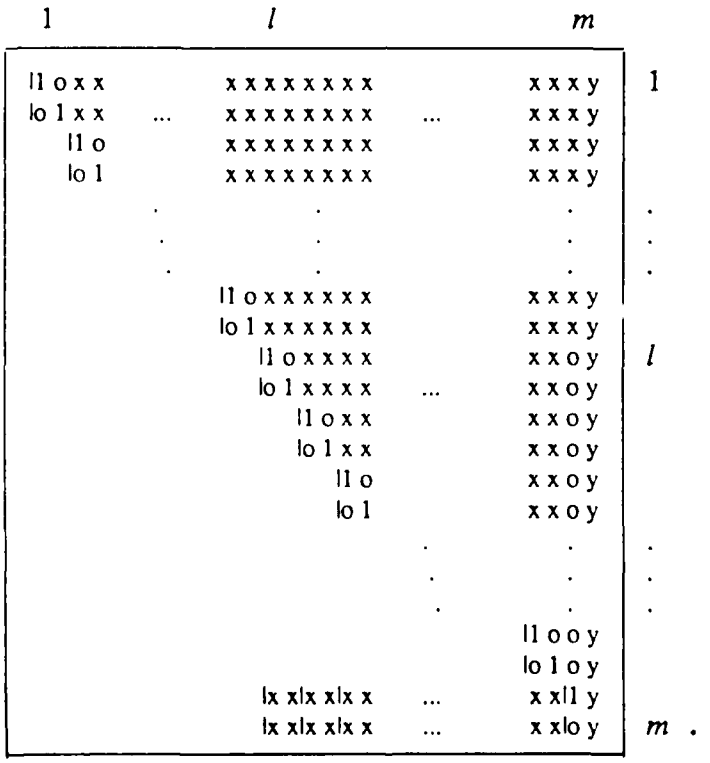


Figure 3.2. Illustration of the general form of the matrix H^l.

$$= \hat{R}^{-1} H^{q}$$
$$= H^{\bar{q}}. \tag{3.6}$$

Consider the matrix H^{l} whose general form is depicted in Figure 3.2. Note that the matrix resulting from the above permutation is H^{l} when $l = \bar{q}$. Note also that all non-W-elements in H^{l} are in the last two rows in columns l through m. Our objective now is to reduce $H^{\bar{q}}$ to a LRQ matrix by eliminating these non-W-elements. We consider eliminating them four at a time using the 2x2 identities on the diagonal of H^{l}. The necessary matrices that should reduce H^{l} to H^{l+2}, for $l=\bar{q},\bar{q}+2,\cdots,m-3$, are the following EULQ transformations.

$$Z^{l} = \begin{array}{c|cc|c} & l-1 & l & \\ \hline & 0 & 0 & \\ \mathrm{I} & \vdots & \vdots & \\ & 0 & 0 & \\ \hline & 1 & 0 & \\ & 0 & 1 & \\ \hline & 0 & 0 & \\ & \vdots & \vdots & \mathrm{I} \\ & 0 & 0 & \\ \hline & -H^{l}_{m-1,l-1} & -H^{l}_{m-1,l} & \\ & -H^{l}_{m,l-1} & -H^{l}_{m,l} & \end{array}$$

By repetitive application of Z^{l} to H^{l}, for $l = \bar{q},\bar{q}+2,\cdots,m-3$, we get H^{m-1} which, in general, has a non-W-element in its $m-1,m$ entry and a nonconforming element in the m,m^{th} entry. Therefore, the following rank-one elementary transformation is sufficient to reduce H^{m-1} to the LRQ matrix $\overline{W}$,

$$Z^{m-1} = \begin{array}{c|c} \multicolumn{2}{c}{\overset{m}{}} \\ \hline \mathrm{I} & \\ \hline & \begin{array}{c} -H^{m-1}_{m-1,m} / H^{m-1}_{m,m} \\ 1 / H^{m-1}_{m,m} \end{array} \end{array} \begin{array}{l} \\ \hline m-1 \\ m \end{array}$$

Theoretically, $H^{m-1}_{m,m}$ is a nonzero element, since otherwise $\bar{B}$ is singular. Now, combining all transformations applied to $H^{\bar{q}}$, we obtain,

$$Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} H^{\bar{q}} = \bar{W},$$

and (3.6) becomes,

$$\{Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} \hat{R}^{-1} Z^{-1}\} \bar{B} \{R^{-1} \hat{R}\} = Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} H^{\bar{q}} \qquad (3.7)$$

$$\{\bar{Z}^{-1}\} \bar{B} \{\bar{R}^{-1}\} = \bar{W},$$

which is equivalent to the required updated form (3.4), with

$$\bar{Z} = Z \hat{R} Z^{\bar{q}^{-1}} \cdots Z^{m-3^{-1}} Z^{m-1^{-1}}, \qquad (3.8)$$

$$\bar{R} = R^{-1} \hat{R}, \text{ and}$$

$$\bar{W} = Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} H^{\bar{q}}.$$

Note that $\bar{Z}$ in (3.8) is not a ULQ matrix, even though all its factors, except the permutation matrix $\hat{R}$, are ULQ matrices. In practice, $\bar{Z}^{-1}$ is stored factorized as in the first braced term in the left hand side of (3.7).

Using the above, the updating algorithm may be stated as follows:

Algorithm 3.1 : The B.Q.I.F. Updating Algorithm

0. Begin with the m x m matrix $B = Z W R$, and suppose column p of B is replaced by $A_{\cdot j}$.

1. Define q such that $R_{\cdot p} = e_q$.

2. Set

$$S_{.,i} \leftarrow \begin{cases} Z^{-1} A_{.,j}, & i = q; \\ W_{.,i}, & \textit{otherwise}. \end{cases}$$

3. Let

$$\hat{q} = \begin{cases} q+1, \text{ if } q \textit{ is odd}; \\ q-1, \text{ if } q \textit{ is even}. \end{cases}$$

4. Set

$$\hat{R}_{.,i} \leftarrow \begin{cases} e_i, & 1 \le i < q; \\ e_{i+2}, & q \le i < m-1; \\ e_{\hat{q}}, & i = m-1; \\ e_q, & i = m. \end{cases}$$

5. $H \leftarrow \hat{R}^{-1} S \hat{R}$.

6. Let

$$\bar{q} = \begin{cases} q, \text{ if } q \textit{ is odd}; \\ q-1, \text{ if } q \textit{ is even}. \end{cases}$$

For $l = \bar{q}, \bar{q}+2, \cdots, m-3$.

7. Set

$$Z^l_{i,l} \leftarrow \begin{cases} 1, & i = l; \\ -H_{m-1,l}, & i = m-1; \\ -H_{m,l}, & i = m; \\ 0, & \textit{otherwise}. \end{cases}$$

$$Z^l_{i,l+1} \leftarrow \begin{cases} 1, & i = l+1; \\ -H_{m-1,l+1}, & i = m-1; \\ -H_{m,l+1}, & i = m; \\ 0, & \textit{otherwise}. \end{cases}$$

$$Z_{.,j} \leftarrow e_j, \quad j \neq l \text{ and } j \neq l+1.$$

8. $H \leftarrow Z^l H$.

Next l.

9. Set

$$Z_{i,m}^{m-1} \leftarrow \begin{cases} -H_{m-1,m}/H_{m,m}, & i=m-1; \\ 1/H_{m,m}, & i=m; \\ 0, \textit{otherwise}. \end{cases}$$

$$Z_{\cdot,j}^{m-1} \leftarrow e_j, \; j \neq m.$$

10. $H \leftarrow Z^1 H$.

11. Set

$$\bar{B} = \{Z \hat{R} (Z^{\bar{q}})^{-1} \cdots (Z^{m-1})^{-1}\} H \{\hat{R}^{-1} R\}$$

$$= \bar{Z} \bar{W} \bar{R}.$$

This updating scheme inherits the major characteristics of the Forrest-Tomlin update for the triangular factors of the basis. First, no new nonzeros are created in the right factor W, since only deletions of items are required. Therefore, sparsity of W is preserved and fill-in is minimized. Second, the lack of choice of the pivot elements makes this update less numerically stable than the Bartels-Golub-based updates. Thus, there is a gain in speed and storage at some sacrifice in numerical stability.

IV. PARALLEL IMPLEMENTATION

In this section we describe a parallel implementation of two basic tasks of any simplex based linear programming code, namely, basis reinversion and solution of the linear systems. A parallel version of the Backward Quadrant Interlocking Factorization Algorithm (BQIF) is presented in Section 4.1. Only the left factor is produced in its product form while the right factor is produced in its explicit form. This form conforms with the updating scheme of Section III. In this algorithm, parallelism is gained by reformulating the BQIF Algorithm in terms of high-level modules such as matrix-vector operations. These modules represent a high level of granularity in the algorithm in the sense that they are based on matrix-vector operations, $O(m^2)$ work, not just vector operations, $O(m)$ work. The module concept has already proven to be very successful in achieving both transportability and high performance of some linear algebra routines across a wide range of architectures, as reported by Dongarra and Sorensen [1984-2] and Dongarra and Hewitt [1986-1].

Given a basic feasible solution with basis B, each iteration of Dantzig's simplex algorithm involves solving the systems of equations $\pi B = c^B$ and $B\,y = A_{\cdot j}$. An efficient parallelization of the simplex algorithm requires efficient parallel algorithms for solution of these systems. Parallel algorithms for solving these linear systems using the quadrant factors are presented in Section 4.2. The parallel implementation discussed in this section is proposed for an MIMD parallel computer that incorporates p identical processors sharing a common memory and capable of *multitasking*, that is, the processors are capable of applying all their power to a single job in a timely and coordinated manner.

4.1 The Module-Based BQIF Algorithm

Given an m x m matrix B, the algorithm either indicates singularity of B or produces

$$Z^{m-1} Z^{m-3} \cdots Z^1 B R = W, \tag{4.1}$$

where R is a permutation matrix, Z^k is a rank-2 matrix of the form,

$$Z^k = \begin{array}{c|cc|c} & k & k+1 & \\ \hline I & & & \\ \hline & x & x & k \\ & x & x & k+1 \\ \hline & x & x & \\ & \cdot & \cdot & \\ & \cdot & \cdot & I \\ & \cdot & \cdot & \\ & x & x & \\ \hline \end{array} , \tag{4.2}$$

This form conforms with the updating schemes of Section III. Its LU version has been used in several LP codes (Reid [1982-1]) . At every stage a new Z^i is produced and two rows of W are updated. The availability of the updated rows of W at every stage allows for parallel implementation when searching for a nonsingular 2x2 submatrix. Moreover, it facilitates finding the 2x2 submatrix of largest determinant rather than finding one with a nonzero determinant. This reduces the rounding error in the factorization process and hence improves the numerical accuracy of the results (Shanehchi and Evans [1982-1]).

The major part of the algorithm is formulated in terms of three basic modules:

Module 1 : Search for a nonsingular 2x2 submatrix

Input : $A \in R^{2,n}$

Purpose : Find column indices j_1 and j_2 such that

$$DET = A_{1,j_1} . A_{2,j_2} - A_{2,j_1} . A_{1,j_2} \neq 0.$$

Output : j_1, j_2, and DET or a singular indication.

Module 2 : Matrix - 2 vectors product

Input : $y^1 \in R^{n_1,2}, A^1 \in R^{n_1,n_2}, x^1 \in R^{n_2,2}$.

Purpose : Compute y^1 such that $y^1 \leftarrow y^1 + A^1 x^1$.

Output : y^1.

Module 3 : 2 vectors - matrix product

Input : $y^2 \in R^{2,l_2}, x^2 \in R^{2,l_1}, A^2 \in R^{l_1,l_2}$.

Purpose : Compute y^2 such that $y^2 \leftarrow y^2 + x^2 A^2$.

Output : y^2.

These modules represent a high level of granularity in the algorithm in the sense that they are based on matrix-vector operations, $O(m^2)$ work, not just vector operations, $O(m)$ work.

Algorithm 4.1 : Reinversion

Let $B \in R^{m,m}$. Then the following steps produce a singular indication if B is singular, or decompose B as in (4.1) if B is nonsingular. The column indices are stored in $IPVT(m)$. Define the 2x2 submatrix $S_{i,j}(A)$ to be

$$S_{i,j}(A) = \begin{bmatrix} A_{i,j} & A_{i,j+1} \\ A_{i+1,j} & A_{i+1,j+1} \end{bmatrix}. \tag{4.3}$$

0. Initialize.

$$W_{1:2,1:m} \leftarrow B_{1:2,1:m}$$

For $i = 1,3,\cdots,m-3$

1. Find a nonsingular 2x2 submatrix.

Set $n \leftarrow m - i + 1$ and $A \leftarrow W_{i:i+1,i:m}$.

Call Module 1 (A, n).

If A is singular, then terminate with B singular;

otherwise, permute columns

$W_{1:m,i}$ with $W_{1:m,j_1}$ and $W_{1:m,i+1}$ with $W_{1:m,j_2}$.

Record permutation, $IPVT(i)=j_1$, $IPVT(i+1)=j_2$.

2. Obtain a new Z^i.

$Z^i \leftarrow I$, where I is m x m, $S_{i,i}(Z^i) \leftarrow [S_{i,i}(W)]^{-1}$.

$n_1 \leftarrow m-i-1$, $n_2 \leftarrow i-1$.

$y^1 \leftarrow B_{i+2:m,i:i+1}$, $x^1 \leftarrow W_{1:i-1,i:i+1}$.

For $l = 1,3,\cdots,i-2$

$A^1_{:,l:l+1} \leftarrow Z^l_{i+2:m,l:l+1}$.

Next l.

Call Module 2 (y^1,x^1,A^1,n_1,n_2)

$Z^i_{i+2:m,i:i+1} \leftarrow y^1$.

3. Update rows $i+2$, $i+3$ of W.

$l_1 \leftarrow i+1$, $l_2 \leftarrow m-i+1$.

$A^2 \leftarrow W_{1:i+1,i+2:m}$, $y^2 \leftarrow B_{i+2:i+3,i+2:m}$.

For $l=1,3,\cdots,i$

$x^2_{:,l:l+1} \leftarrow Z^l_{i+2:i+3,l:l+1}$.

Next l.

Call Module 3 (y^2,x^2,A^2,l_1,l_2)

$W_{i+2:i+3,i+2:m} \leftarrow y^2$.

Next i.

4. Update W.

For $i =1,3,\cdots,m-3$

$S_{i,i}(W) \leftarrow I$, where I is 2 x 2.

$W_{i:i+1,i+2:m} \leftarrow S_{i,i}(Z^i)\, W_{i:i+1,i+2:m}$

Next i.

The general approach we propose for parallel implementation involves having the parent processor prepare the parameters for a module and make use of the kids (subtask processors) to work concurrently on that module. In Module 1, at most $n(n-1)/2$ column pairs should be checked. The parent sends to each kid the column indices to be checked for nonsingularity, and stops all kids whenever one succeeds. As mentioned before, it is possible to find the nonsingular 2x2 submatrix of largest determinant. To do this, the parent sends the column indices to the kids, each kid finds the column pair of largest determinant in his list, sends them to the parent, then the parent selects the best by comparing only $p-1$ values.

The concurrency in Modules 2 and 3 is obvious since they involve matrix-vector operations. In Module 2 (matrix - 2 vectors product) parallelism is obtained by performing $2n_1$ independent inner products, where n_1 is the row dimension of the matrix. Similarly, in Module 3 (2 vectors - matrix product) concurrency is gained by executing $2l_2$ independent inner products, where l_2 is number of columns of the matrix. Step 3 needs only $Z^i_{i+2,i+3,i,i+1}$ from Step 2. These are the first two rows of y^1. Thus, as soon as these elements become available Step 3 may proceed. This can easily be synchronized. Finally, in Step 4 the loop divides over i with completely independent tasks. However, the tasks require different amounts of computation. Two solutions are possible. Either we adopt dynamic task queue allocation, or we statically allocate $i=1, m-3$ to one processor, $i=3, m-5$ to the second, and so on.

4.2 Solving the Linear Systems

In this section we investigate the possible parallelism involved when we solve the systems of equations $\pi B = c^B$ and $B\ y = A_{.j}$. We assume that the basis matrix B is in the form (4.1), that is

$$Z^{m-1} Z^{m-3} \cdots Z^1 B R = W,$$

where Z^k has the form (4.2), and W is a block unit upper triangular matrix with blocks of size 2,that is it has the form (2.36). We compute the dual variables (π) using the following steps:

(1) Permutation : $\bar{\pi} = c^B R$.

(2) Solve a block triangular system : $\hat{\pi} W = \bar{\pi}$.

(3) BTRAN : $\pi = \hat{\pi} Z^{m-1} Z^{m-3} \cdots Z^1$.

We compute y, the basis representation of the incoming column $A_{.j}$, as follows:

(1) FTRAN : $\hat{y} = Z^{m-1} Z^{m-3} \cdots Z^1 A_{.j}$.

(2) Solve a block triangular system : $W \bar{y} = \hat{y}$.

(3) Permutation : $y = R \bar{y}$.

We present parallel implementations of the FTRAN operation, the solution of a block triangular system, and the BTRAN operation in Sections 4.2.1, 4.2.2, and 4.2.3, respectively.

4.2.1 The FTRAN in Parallel

The rules for applying a Z^k to an arbitrary vector v are as follows:

a) Extract $\alpha_k \leftarrow v_k$, and $\alpha_{k+1} \leftarrow v_{k+1}$.

b) Set $v_k \leftarrow 0$, and $v_{k+1} \leftarrow 0$.

c) Compute $\bar{v} = v + \alpha_k Z^k_{:m,k} + \alpha_{k+1} Z^k_{:m,k+1}$.

Note that if $v_k = v_{k+1} = 0$, then $\bar{v} = v$ and no element of v will change.

An example is now given for $m = 6, k = 3$. Suppose we have

$$Z^3_{:3:4} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 1 & 2 \\ 1/3 & 1/4 \\ 1/6 & 1/2 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}, \text{ and } \quad u = \begin{bmatrix} 11 \\ 12 \\ 0 \\ 0 \\ 15 \\ 16 \end{bmatrix}.$$

Then the computation of $Z^3 v$ is given by

$$Z^3 v = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \\ 5 \\ 6 \end{bmatrix} + 3 \begin{bmatrix} 0 \\ 0 \\ 2 \\ 1 \\ 1/3 \\ 1/6 \end{bmatrix} + 4 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 1/4 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 10 \\ 11 \\ 7 \\ 8.5 \end{bmatrix},$$

and the computation of $Z^3 u$ is given by

$$Z^3 u = \begin{bmatrix} 11 \\ 12 \\ 0 \\ 0 \\ 15 \\ 16 \end{bmatrix} + 0 \begin{bmatrix} 0 \\ 0 \\ 2 \\ 1 \\ 1/3 \\ 1/6 \end{bmatrix} + 0 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 1/4 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \\ 0 \\ 0 \\ 15 \\ 16 \end{bmatrix}.$$

These rules are implemented in the following module:

Module : FTRAN Operation (A, v, n)

Purpose : Apply Z^k to an arbitrary vector v.

Input : n, $A \in R^{n,2}$, $v \in R^{n,1}$.

Output : v, where $v = Z^k v$.

Steps : 1. Extract $\alpha_1 \leftarrow v_1$, and $\alpha_2 \leftarrow v_2$.

2. Set $v_1 \leftarrow 0$, and $v_2 \leftarrow 0$.

3. Compute $A_{.,1} \leftarrow \alpha_1 A_{.,1}$.

4. Compute $A_{.,2} \leftarrow \alpha_2 A_{.,2}$.

5. Compute $v \leftarrow v + A_{.,1} + A_{.,2}$.

Obviously, steps 3 and 4 are independent and can be executed in parallel. In step 5, the work is partitioned over the rows of v, assigning each kid a block of rows to evaluate.

4.2.2 Solving the Block Triangular System

The solution of an m x m triangular system of equations on a sequential computer can be obtained by either a forward or backward substitution process which requires $O(m^2)$ steps, each defined as one multiplication followed by one addition. In order to solve the system on a parallel computer, methods which require $O(m^3)$ processors and, hence, reduce the computation time to $O(\log^2 m)$ have been developed (e.g. Chen and Kuck [1975-1] and Sameh and Brent [1977-1]). Evans and Dunbar [1983-1] introduced methods that run in $O(m)$ time using $O(m)$ processors. For practical purposes the processor and storage requirement of these methods is unreasonably large.

In this subsection we consider solving the linear system

$$xW = b, \tag{4.4}$$

where $x, b \in R^m$ and W is an upper triangular m x m matrix with 2x2 identity diagonal blocks. This system may be solved by a forward substitution (FS) process described in algorithmic form as follows.

For $i = 1,2,\cdots,m$

$$x_i = b_i - \sum_{j=1}^{i-1} W_{i,j}\, x_j .$$

Next i.

It is obvious that a uniprocessor will solve (4.4) sequentially in $m(m-2)/2$ steps by the FS process. Let T_p denote the time required to solve (4.4) using p processors, where one step requires one unit of time. Then

$$T_1 = m\,(m-2)/2.$$

With a parallel computer that has p processors, a minimum time requirement for the solution of (4.4) is

$$\min(T_p) = T_1/p = m\,(m-2)/(2p). \tag{4.5}$$

The minimum completion time of any algorithm based on FS is equal to the number of terms in the expression that evaluates x_m, that is

$$T_{\min} = m - 2.$$

From (4.5) it is clear that a minimum of $m/2$ processors is necessary to solve (4.4) in the minimum time of $m-2$ operations. Again this processor requirement is unreasonably large for our application.

The machine we consider has a limited number of identical processors ($p \leq 30$). Therefore, we consider the question: if we are given a fixed number of processors, how should the parallel operations be scheduled on the processors to minimize the solution time of (4.4)? We propose to answer this question using a directed graph model that represents the FS process as follows. The nodes of the graph represent tasks of equal execution time and the edges represent the precedence relationships between the tasks. Then we apply a simple scheduling algorithm due to Hu [1961-1], called the *level algorithm*, to schedule the tasks on the processors such that the total execution time is minimized. This algorithm is known to be optimum for a tree graph, and it gives extremely good results for general graphs as reported by Ramamoorthy et al [1972-1], Huang and Wing [1979-1], and Wing and Huang [1980-1].

We first organize the FS process in terms of operations of equal time and define the corresponding directed graph. Let $x^i = [x_i, x_{i+1}]$. Partition x, b, and W into blocks of size 2. Using $S_{i,j}$ as defined in (4.3), the above FS process can then be written as

For $i = 1,3, \cdots, m-1$

$$x^i = b^i - \sum_{j=1,3,\cdots,i-2} x^j \, S_{i,j}(W).$$

Next i.

Let the following operation, where x^i is used to update x^j, define a task

$$x^j \leftarrow x^j - x^i \, S_{i,j}(W). \tag{4.5}$$

For Hu's algorithm we assume that the execution time of an operation (4.5) is one unit (4 multiplications and 4 additions). We can see that the FS process consists of a set of operations (4.5), on which a set of precedence relations exists. That is, to complete the evaluation of x^i we require x^{i-2}, for $i = 3,5,\cdots,m-1$. The process can therefore be represented by a directed graph $G(V,E)$ where the vertex set V is defined as

$$V \equiv \{v_{i,j} \mid v_{i,j} \textit{ represents an operation } (4.6)\},$$

and the edge set E is defined as

$$E \equiv \{(v_{i,j}, v_{k,l}) \mid \textit{operation } v_{k,l} \textit{ requires the direct result of operation } v_{i,j}\}.$$

We shall call $G(V,E)$ the *forward substitution task graph,* and refer to it by FSTG. In Figure 4.1 the FSTG for m=10 is presented. For every $v_{i,j}$ in the FSTG, the pair i,j is indicated. A node is an *initial node* if it does not have a predecessor and is a *terminal node* if it has no successor. It is clear that the FSTG has only one terminal node, at which $i = m-3$ and $j = m-1$. Accordingly, the minimum completion time, denoted by D, of the FSTG is equal to the number of nodes on the longest path from an initial node to the terminal node. Thus, $D = (m/2) - 1$, which is the number of times operation (4.6) is executed for x^{m-1}.

We next determine the levels of the vertices of the FSTG. Define the *level number* ($l_{i,j}$) of a node $v_{i,j}$ as follows: 1) the level of the terminal node is D, 2) the level of a node that has one or more successors is equal to the minimum of the levels of its successors minus one. Applying this definition to the FSTG, we can conclude that

$$l_{i,j} = (i + 1) / 2. \qquad (4.6)$$

The level number is simply the latest time by which node $v_{i,j}$ must be processed in order to complete the task graph in the minimum time D. The level numbers of the nodes of Figure 4.1 are given as shown.

Once the level numbers of the operations are determined, we apply Hu's scheduling

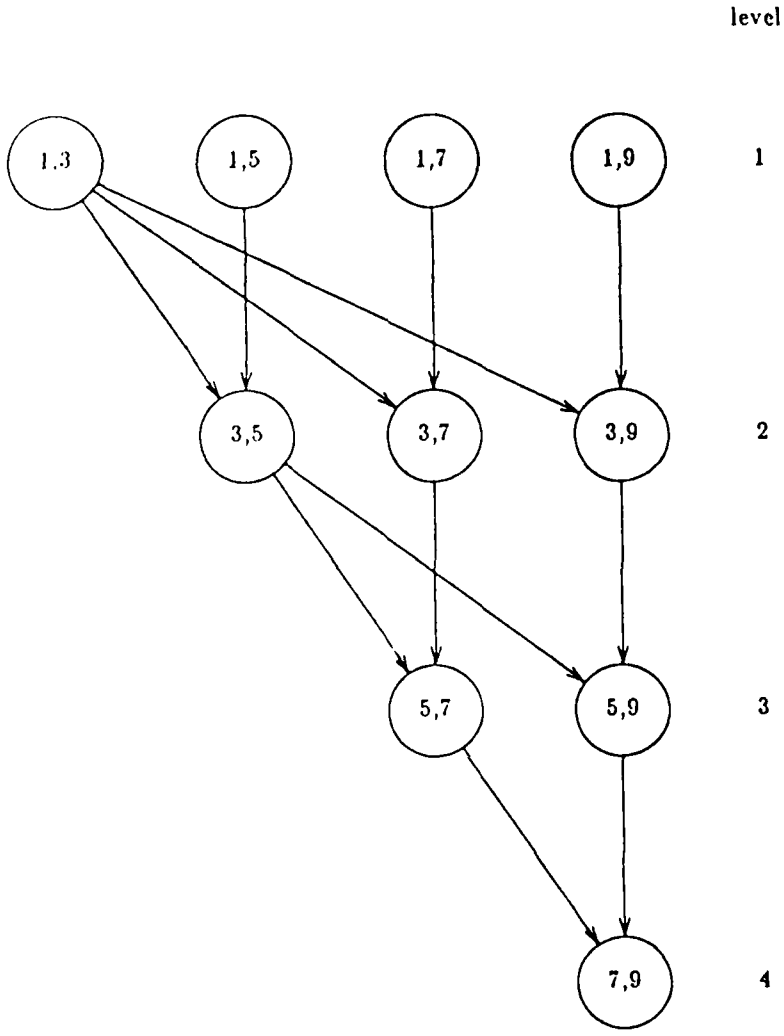


Figure 4.1. The FSTG for $m = 10$.

algorithm to assign operations to processors. Define a *ready task* to be one whose immediate predecessors have all been processed. The scheduling algorithm is as follows.

Algorithm 4.2: Hu's Scheduling Algorithm

1. Among all the ready tasks, schedule the one with smallest level number.

2. If there is a tie, schedule the one with the largest number of immediate successors.

Applying this Algorithm to the FS process represented by FSTG, the computations are organized as follows.

Algorithm 4.3 : Forward Substitution

Set $x^1 \leftarrow b^1$.

For $k = 3,5,\cdots,m-1$

$x^k \leftarrow b^k - x^1 S_{1,k}(W)$.

Next k.

For $i = 3,\cdots,m-3$.

For $j = i+2, i+4,\cdots,m-1$

$x^j \leftarrow x^j - x^i S_{i,j}(W)$.

Next j.

Next i.

All operations in loop k are independent and have the same level number. Their level number ($l_{1,k} = 1$) is the smallest among all other operations in the Algorithm, and hence they are executed first. Similarly, all operations in loop j are independent and have the same level number as given by (4.6). The ordering of index i predicates the execution of the operations by increasing level number. This satisfies the first criterion in Hu's Algorithm. The second criterion imposes the ordering of the index j. That is, the number of immediate successors of $v_{i,j}$ is always greater than or equal to that of $v_{i,j+2}$ for

$j = i+2, i+4, \cdots, m-1$.

A parallel implementation of Algorithm 4.3 involves having the parent processor partition the work in loop k among the kids. Then for every i, the computational tasks of loop j are again divided among the kids.

Lower bounds on the completion time of a task graph given a fixed number of processors were derived by Ramamoorthy et al. [1972-1]. Let n_k be the number of nodes in level k. Let $t^*(p)$ be the minimum completion time to process a task graph with p processors. Then

$$t^*(p) \geq \max_i \left\lceil \frac{\sum_{k=1}^{i} n_k}{p} + D - i \right\rceil, \tag{4.7}$$

where D is the minimum completion time of the task graph and $\lceil x \rceil$ denotes the smallest integer $\geq x$. The first term in the expression denotes the minimum number of time units required to complete all the operations of the first i levels using p processors. The term $D-i$ is equal to the number of remaining levels yet to be processed. This bound may be useful in demonstrating optimality of the scheduling using Hu's Algorithm.

4.2.3 Parallel Implementation of the BTRAN Operation

In this section, we consider the parallel implementation of the following operation

$$\pi = \hat{\pi} Z^{m-1} Z^{m-3} \cdots Z^1,$$

where $\hat{\pi}$ is an arbitrary vector of m elements and each Z^k is an m x m rank-2 matrix that has the form (4.3).

The rule for computing $\bar{u} = u\ Z^k$ is as follows:

a) Set $\bar{u}_i \leftarrow u_i$ for $i \neq k$ and $i \neq k+1$.

b) Set $\bar{u}_k \leftarrow u_{k:m}\ Z^k_{k:m,k}$.

c) Set $\bar{u}_{k+1} \leftarrow u_{k:m}\ Z^k_{k:m,k+1}$.

For example, let $m = 6, k = 3$ and suppose we have

$$Z^{3}_{:,3:4} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 1 & 2 \\ 3 & 4 \\ 6 & 2 \end{bmatrix}, \text{ and } \quad u = [\,1\ 1\ 1\ 1\ 1\ 1\,].$$

Then $\bar{u} = u\ Z^{k} = [\,1\ \ 1\ \ 12\ \ 9\ \ 1\ \ 1\,]$.

Note that $\bar{u}$ differs from u in only the k^{th} and the $k+1^{st}$ elements. Note also that the elements $u_i, i=1, \cdots, k-1$, are not required in computing $\bar{u}$. Using these observations, the BTRAN process may be represented by the following.

For $k = m-1, \cdots, 1$

$u_k \leftarrow u_{k:m}\ Z^{k}_{k:m,k}$.

$u_{k+1} \leftarrow u_{k:m}\ Z^{k}_{k:m,k+1}$.

Next k.

We now apply the methodology stated at the end of the previous subsection. Let the following operations define a task

$$u^{k} \leftarrow u^{k}\ S_{k,k}(Z^{k}). \tag{4.8}$$

$$u^{j} \leftarrow u^{j} + u^{i}\ S_{i,j}(Z^{j}). \tag{4.9}$$

We assume that the execution time of both operations is one unit. The task graph $G(V,E)$ of the BTRAN process is defined by the vertex set V, where

$$V \equiv \left\{ v_{i,j} \mid v_{i,j}\ \textit{represents} \left\{ \begin{array}{l} \textit{an operation (4.8), if } i=j; \\ \textit{an operation (4.9), otherwise} \end{array} \right\} \right\},$$

and the edge set E, where

$$E \equiv \{(v_{i,j}, v_{k,l}) \mid \textit{operation } v_{k,l} \textit{ requires the direct result of operation } v_{i,j}\}.$$

$G(V',E)$ has only one terminal node at which $i = 3$ and $j = 1$. Following the same arguments used earlier with FSTG, we conclude that

$$D = m / 2,$$

and

$$l_{i,j} = \begin{cases} 1, \text{if } i = j; \\ (m - i + 3) / 2, \textit{otherwise}. \end{cases}$$

Applying Hu's Algorithm to the BTRAN task graph yields the following ordering of computations.

Algorithm 4.4 : BTRAN Operation

For $k = m-1, m-3, \cdots, 1$

  $u^k \leftarrow u^k \, S_{k,k}(Z^k)$.

Next k.

For $i = m-1, m-3, \cdots, 3$

  For $j = i-2, i-4, \cdots, 1$

    $u^j \leftarrow u^j + u^i \, S_{i,j}(Z^j)$.

  Next j.

Next i.

The ordering of the index i is imposed by the first criterion of Hu's Algorithm. The ordering of the indices k and j is the result of applying the second criterion. Parallelism is gained by having the kid processors work first on loop k in parallel, and then for every i, having the kid processors work on loop j in parallel.

V. SUMMARY

Evans and Hatzopoulos [1979-1] developed a new matrix factorization, known as *the Quadrant Interlocking Factorization (QIF),* for solving linear systems on parallel computers. In this paper we have presented the algorithms required to use this new factorization in Dantzig's simplex algorithm for linear programming. This work may be viewed as a parallelization of the simplex method using a quadrant interlocking factorization for the basis inverse.

In Section II, the factorization algorithms are developed, and the relationship of quadrant and triangular matrices is presented. In Section III, a new algorithm is presented for updating the factorization during a basis exchange step. In Section IV, we present a parallel implementation of the factorization algorithm, and develop the algorithms required to solve the linear systems of the simplex method on a parallel computer using the QIF of the basis. For each algorithm the concurrency among the steps is revealed, the computations are organized and a parallel implementation is proposed. The algorithms are designed for an MIMD parallel computer that incorporates p identical processors sharing a common memory and capable of applying all their power to a single application in a timely and coordinated manner.

REFERENCES

Bartels, R. H., 1971-1, "A Stabilization of the Simplex Method," Numer. Math., 16, pp. 414-434.

Chen, S. C., and D. Kuck, 1975-1, "Time and Parallel Processor Bounds for Linear Recurrence Systems," IEEE Trans. Comput., C-24, pp. 101-117.

Chen, S. S., J. J. Dongarra and C. C. Hsiung, 1984-1, "Multiprocessing Linear Algebra Algorithms on the CRAY X-MP-2: Experiences with Small Granularity," J. Parallel and Distributed Computing, 1, pp. 22-31.

Dongarra, J. J., A. H. Sameh and D. C. Sorensen, 1984-1, "Implementation of Some Concurrent Algorithms for Matrix Factorization," Argonne Nat. Lab., Argonne, IL, Rep. ANL/MCS-TM-25.

_____, and D. C. Sorensen, 1984-2, "A Parallel Linear Algebra Library for the Denelcor HEP," Argonne Nat. Lab., Argonne, IL, Rep. ANL/MCS-TM-33.

_____, and T. Hewitt, 1986-1, "Implementing Dense Linear Algebra Algorithms Using Multitasking on the CRAY X-MP-4," SIAM J. Sci. Stat. Comput., 7, pp. 347-350.

Evans, D. J., and M. Hatzopoulos, 1979-1, "A Parallel Linear System Solver," Intern. J. Computer Math., 7, pp. 227-238.

_____, and A. Hadjidimos, 1980-1, "A Modification of the Quadrant Interlocking Factorisation Parallel Method," Intern. J. Computer Math., 8, pp. 149-166.

_____, 1982-1, "Parallel Numerical Algorithms for Linear Systems," in Parallel Processing Systems, (D. J. Evans, ed.), Cambridge Univ. Press, Cambridge , pp. 357-384.

_____, and R. C. Dunbar, 1983-1, "The Parallel Solution of Triangular Systems of Equations," IEEE Trans. Comput., C-23, pp. 201-204.

Feilmeier, M., 1982-1, "Parallel Numerical Algorithms," in Parallel Processing Systems, (D. J. Evans, ed.), Cambridge University Press, Cambridge , pp. 285-338.

Forrest, J. J. H., and J. A. Tomlin, 1972-1, "Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method," Mathematical Programming, 2, pp. 263-278.

Hellier, R. L., 1982-1, "DAP Implementation of the WZ Algorithm," Comp. Phys. Comm., 26, pp. 321-323.

Huang, J. W., and O. Wing, 1979-1, "Optimal Parallel Triangulation of a Sparse Matrix," IEEE Trans. Circuits Syst., CAS-26, pp. 726-732.

Hu, T. C., 1961-1, "Parallel Sequencing and Assembly Line Problems," Operations Research, 9, pp.841-848.

Markowitz, H. M., 1957-1, "The Elimination Form of the Inverse and its Application to Linear Programming," Management Science, 3, pp. 255-269.

Ramamoorthy, C. V., K. M. Chandy and M. J. Gonzalez, 1972-1, "Optimal Scheduling Strategies in a Multiprocessor System," IEEE Trans. Comput., C-21, pp. 137-146.

Reid, J. K., 1982-1, "A Sparsity Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases," Math. Programming, 24, pp. 55-69.

Sameh, A. H., and R. P. Brent, 1977-1, "Solving Triangular Systems on a Parallel Computer," SIAM J. Numer. Anal., 14, pp. 1101-1113.

Saunders, M. A., 1976-1, "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating," in Sparse Matrix Computations, (J. R. Bunch and D. J. Rose, eds.), Academic Press, New York, New York, pp. 213-226.

Shanehchi, J. and D. J. Evans, 1982-1, "Further Analysis of the Quadrant Interlocking Factorisation (Q.I.F.) Method," Intern. J. Computer Math., 11, pp. 49-72.

Wing, O., and J. W. Huang, 1980-1, "A Computation Model for Parallel Solution of Linear Equations," IEEE Trans. Comput., C-29, pp. 632-638.

Zaki, H. A., 1986-1, "A Parallelization of the Simplex Method Using the Quadrant Interlocking Factorization," unpublished dissertation, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas.

CHAPTER 7

Technical Report 87-OR-02

MINIMAL SPANNING TREES:

A COMPUTATIONAL INVESTIGATION OF PARALLEL ALGORITHMS

by

R. S. Barr

R. V. Helgason

and

J. L. Kennington

Department of Operations Research
School of Engineering and Applied Scienc
Southern Methodist University
Dallas, Texas 75275

July 1987

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

The objective of this investigation is to computationally test parallel algorithms for finding minimal spanning trees. Computational tests were run on a single processor using Prim's, Kruskal's and Boruvka's algorithms. Our implementation of Prim's algorithm is superior for high density graphs, while our implementation of Boruvka's algorithm is best for sparse graphs. Implementations of parallel versions of both Prim's and Boruvka's algorithms were tested on a twenty-cpu Balance 21000. For the environment in which a minimum spanning tree problem is a subproblem within another algorithm, the parallel implementation of Boruvka's algorithm produced speedups of three and five on five and ten processors, respectively; while the parallel implementation of Prim's algorithm produced speedups of three and five on five and ten processors, respectively. The one-time overhead for process creation negates most, if not all of the benefits for solving a single minimum spanning tree subproblem.

ACKNOWLEDGEMENT

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C-0182, the Air Force Office of Scientific Research under Contract Numbers AFOSR 83-0278 and AFOSR 87-0199, the Office of Naval Research under Contract Number N00014-87-K-0223, and ROME Air Development Center under Contract Number SCEEE PDP/86-75. The authors wish to express their appreciation to Professor Hossam Zaki of the University of Illinois and Professor Iqbal Ali of the University of Massachusetts at Amherst for their helpful comments.

I. INTRODUCTION

The United States along with other developed countries is entering a new generation of computing that will require software engineers to redesign and reevaluate standard algorithms for the new parallel processing hardware that is being installed throughout the developed world. It may well be that algorithms which proved to be superior for single processor machines may prove to be inferior in some of the new parallel processing environments. One of the more popular new parallel machines is Sequent Computer Systems' Balance 21000. The objective of this investigation is to computationally test parallel algorithms for finding minimal spanning trees on a twenty-cpu Balance 21000.

An undirected graph $G = [V,E]$ consists of a vertex set V and an edge set E. Without loss of generality we assume that the edges are distinct. If $G' = [V',E']$ is a subgraph of G with $V' = V$, then G' is called a spanning subgraph for G. If, in addition, G' is a tree, then G' is called a spanning tree for G. A graph whose components are trees is called a forest, and a spanning subgraph for G, which is also a forest, is called a spanning forest for G. We will call $\{[V_i,T_i]: V_i = \{u_i\}, T_i = \emptyset, u_i \in V\}$ the trivial spanning forest for G and the $[V_i,T_i]$ trivial trees. Associated with each edge (u,v) is a real-valued cost c(u,v). The minimum spanning tree problem may be stated as follows: Given a connected undirected graph each of whose edges has a real-valued cost, find a spanning tree of the graph whose total edge cost is minimum.

Applications include the design of a distribution network in which the nodes represent cities or towns and the edges represent electrical power lines, water lines, natural gas lines, communication links, etc. The objective is to design a network which uses the least length of cable or pipe. The minimum spanning tree problem is also used as a subproblem for algorithms for the travelling salesman problem (see Held and Karp [6, 7] and Ali and Kennington

[3]). Some vehicle routing algorithms require the solution of a travelling salesman problem on a subset of nodes. Hence, a wide variety of applications require the solution of minimal spanning trees. Some applications require a single solution and some use the model as a subproblem within another algorithm.

II. THREE CLASSICAL ALGORITHMS

The algorithms in current use may be traced to ideas developed by Prim, Kruskal, and Boruvka. These three classical algorithms all begin with the trivial spanning forest $G_0 = \{[V_i,T_i],\ i = 0,\ldots,|V|-1\}$. A sequence of spanning forests is obtained by merging spanning forest components. Given spanning forest G_k, a nonforest edge (u,v) is selected and the components $[V_i,T_i]$ and $[V_j,T_j]$ with $u \in V_i$ and $v \in V_j$ are removed from G_k and replaced by $[V_\ell,T_\ell]$, where $\ell = k + |V|$, $V_\ell = V_i \cup V_j$, and $T_\ell = T_i \cup T_j \cup \{(u,v)\}$, yielding spanning forest G_{k+1}. After $m = |V|-1$ edges have been selected, $G_m = \{[V_{2m},T_{2m}]\} = \{[V,T]\}$ is a minimal spanning tree for G.

Let $[V_i,T_i]$ and $[V_j,T_j]$ denote two disjoint subtrees of G. Define d_{ij}, the shortest distance between the trees, by $d_{ij} = \min \{c(u,v):\ (u,v) \in E,\ u \in V_i,\ v \in V_j\}$. The three classical algorithms may be viewed as different applications of the following result:

Proposition 1.

Let $V_0, V_1, \ldots, V_n$ denote vertex sets of disjoint subtrees of a minimum spanning tree for G. Let $c(u,v) = d_{jn} = \min_{j \neq n} d_{jn}$ with $(u,v) \in V_j \times V_n$. Then (u,v) is an edge in a minimal spanning tree for G.

A proof of Proposition 1 may be found in Christofides [4, pp. 135-136].

In Prim's algorithm, the nonforest edge (u,v) for G_k is always selected so that $(u,v) \in V_i \times V_{j*}$, where j^* is the largest index j such that $[V_j,T_j] \in G_k$. Thus a single component continues to grow as trivial trees disappear. An excellent description of Prim's algorithm is given in Papadimitriou and Steiglitz [15, p. 273], along with its (serial) computational complexity of $O(|V|^2)$. It is believed that this algorithm is best suited for dense graphs.

In Boruvka's algorithm, the nonforest edge (u,v) for G_k is always selected so that $(u,v) \in V_{i*} \times V_j$, where i^* is the smallest index i such that $[V_i,T_i] \in$

G_k. Thus a variety of different-sized components may be produced as the algorithm proceeds. All trivial trees will be removed first in the early stages of this algorithm. A description of Boruvka's algorithm is given in Papadimitriou and Steiglitz [15, p. 277], along with its (serial) computational complexity of $O(|E| \log |V|)$. This algorithm appears to be best suited for sparse graphs.

Kruskal's method may be viewed as an application of the greedy algorithm. The minimum spanning tree is constructed by examining the edges in order of increasing cost. If an edge forms a cycle within a component of G_k, it is discarded. Otherwise it is selected and yields G_{k+1}. Here also different-sized components may be produced. A description of Kruskal's algorithm is given in Sedgewick [18, pp. 412-413], along with its (serial) computational complexity of $O(|E| \log |E|)$.

III. COMPUTATIONAL RESULTS WITH SEQUENTIAL ALGORITHMS

Computer codes for Boruvka's algorithm, Kruskal's algorithm, and three versions of Prim's algorithm were developed. SPARSE PRIM maintains the edge data in both forward and backward star format, while DENSE PRIM maintains the edge data in an $|V| \times |V|$ matrix. HEAP PRIM maintains the edge data in both forward and backward star format and makes use of a d-heap as described in Tarjan [19, p. 77]. KRUSKAL makes use of a partial quick sort as described in [1, 8] to produce the least cost remaining edge. BORUVKA is a straightforward implementation of the algorithm presented in [15].

Random problems were generated on both n x n grid graphs and on completely random graphs. All costs were uniformly distributed on the interval [0, maxcost]. All codes are written in FORTRAN for the Balance 21000.

The computational results for grid graphs are presented in Table 1. These graphs are very sparse and BORUVKA was the clear winner. The computational results for random graphs may be found in Tables 2 and 3. SPARSE PRIM was the winner for problems whose density was at least 40% with HEAP PRIM running a close second. For problems with densities of 20% or less, HEAP PRIM was the winner with KRUSKAL running a close second. KRUSKAL appeared to be the most robust implementation, working fairly well on all problems tested.

Tables 1, 2, 3 About Here

IV. PARALLEL ALGORITHMS

Parallel versions of the three classical algorithms have appeared in the literature (see [2, 5, 9, 10, 11, 12, 16, 17]), however; no computation experience has been reported. The overhead required for coordinating the work of multiple processors can only be determined by actual implementation on a parallel processing machine.

A parallel version of Boruvka's algorithm was developed for grid graphs and a parallel version of Prim's algorithm was developed for high density random graphs. Both algorithms use modules (subroutines) which may be executed in parallel. Suppose there are p processors available for use. The parallel operations are initiated by the main program using statements of the form:

for m = 1 to p, fork module z(m).

The main program and p-1 clones will each execute module z in parallel. Processing does not continue in the main program until all processors complete module z. The argument "m" allows each of the p processors to process different parts of the data or follow a different path. We assume that all data in the main program is shared with module z. If module z has local non-shared variables, then these will be explicitly stated in the description of the module. Multiple processors which update the same variable, set, or list use locks to insure that only one processor has access to a given item.

4.1 Parallel Boruvka For Grids

Using the fork and lock constructs we present a parallelization of Boruvka's algorithm for grid graphs. The most expensive component of Boruvka's sequential algorithm may be described by the following procedure:

```
for all (u,v) ε E
  let i and j denote the subtrees containing u and v, respectively;
  if i ≠ j then
    if cost(u,v) < min(i) then min(i) ← cost(u,v)
    if cost(u,v) < min(j) then min(j) ← cost(u,v)
  end if
end for
```

That is, all the edge costs must be examined and certain subtree data are updated. Our parallelization of this scan relies upon a partitioning of the grid into p components (one for each processor). A three processor partitioning of a 7 x 7 grid network is illustrated in Figure 1.

Figure 1 About Here

The above edge scan is performed in two stages. The first stage performs a parallel scan over edges both of whose vertices lie within the same partition. The second stage performs a parallel scan over edges across cut sets. If each partition consists of at least two rows of the grid, then all subtree data updating can be performed independently without the requirement of a lock.

The second part of Boruvka's algorithm is to merge two subtrees by appending a new edge. The merger of subtrees, both of which lie in the same partition can also be executed in parallel.

Using this data partitioning approach, the parallel algorithm may be stated as follows:

PARALLEL BORUVKA FOR GRIDS

Input: 1. An n x n grid graph $G = [V,E]$ with $V = \{v_1, \ldots, v_q\}$.

2. For each edge $(u,v) \in E$ a cost $c(u,v)$.

3. The number of processors, p, available for use.

Output: A minimal spanning tree $[V,T]$.

Assumption: G is connected and has no parallel edges.

```
begin
    T ← ∅, r ← ⌈n/p⌉, ℓ ← n - rp;
    If r < 2, terminate.
    for i = 1 to q,  S_i ← {v_i};
    C ← {S_1,..., S_q};
    W_1 ← {v: v ∈ V and v is in grid rows 1 through v + ℓ};
    for m = 2 to p,
       W_m ← {v: v ∈ V and v is in grid rows (m-1)r + ℓ + 1 through mr + ℓ };
    for m = 1 to p, X_1m ← {(u,v): (u,v) ∈ E, u ∈ W_m, and v ∈ W_m};
    for m = 1 to p - 1,
       X_2m ← {(u,v): (u,v) ∈ E with u ∈ W_m, v ∈ W_m+1 or u ∈ W_m+1, v ∈ W_m};
    for i = 1 to q, cpu(i) ← m, where v_i ∈ W_m;
    (comment: S_1,..., S_q are assigned to the p processes)
    create p-1 clones
    (comment: create p-1 additional processes and place them in the wait
              state)
    while |C| ≠ 1
       for m = 1 to p, fork module edgescan(1,m);
       (comment: forks are executed in parallel and processing does not continue
                 in the main program until all processes complete edgescan)
       for m = 1 to p-1, fork module edgescan(2,m);
       L ← ∅;
```

```
        for m = 1 to p, fork module merge(m);
        for all (u,v) ε L do
           let S_i and S_j be the sets containing u and v, respectively;
           if |S_i| < |S_j| then
              S_i ← S_i ∪ S_j, C ← C\S_j;
           else
              S_j ← S_i ∪ S_j, C ← C\S_i;
           end if
           T ← T ∪ (u,v);
        end for
     end while
     kill the clones
end
module edgescan(k,m)
begin
(comment: k = 1 implies the scan is within partition m,
          k = 2 implies the scan is across the cut set separating partitions
                m and m + 1)
        for all (u,v) ε X_km
          let S_i, S_j be the sets containing u and v, respectively;
          if i ≠ j then
            if c(u,v) < min(i) then min(i) ← c(u,v), shortest(i) ← (u,v);
            if c(u,v) < min(j) then min(j) ← c(u,v), shortest(j) ← (u,v);
          end if
          (comment: shortest(i) is the least cost edge incident on S_i)
        end for
end
```

```
module merge(m)
begin
        for all v_k ε W_m do
          (u,v) ← shortest(k)
          let S_i, S_j be the sets containing u and v, respectively;
          if i ≠ j then
            if cpu(i) = cpu(j) then
              if |S_i| < |S_j| then
                S_i ← S_i ∪ S_j, C ← C\S_j;
              else
                S_j ← S_i ∪ S_j, C ← C\S_i;
              end if
              lock T
                T ← T ∪{(u,v)}
              unlock T
            else
              lock L
                L ← L ∪{(u,v)}
              unlock L
            end if
          end if
        end for
end
```

4.2 Parallel Prim

The most expensive part of Prim's sequential algorithm is to find a minimum entry in an $|V|$ length array. This search can be allocated over p processors, each of which finds a candidate minimum. The best of the p candidates becomes the global minimum. Under the assumption that parallel edges do not exist, there is also a scan of edges over the forward and backward star of a given node which can be executed in parallel. Data partitioning via the use of independent cut sets could also be used for random graphs in a manner similar to that described in Section 4.1. That has not been done in this investigation.

The parallelization of Prim's algorithm may be stated as follows:

PARALLEL PRIM

Input: 1. A graph G = [V,E] with $V = \{v_1, \ldots, v_n\}$.

2. For each edge $(u,v) \in E$, a cost c(u,v).

3. The number of processors, p, available for use.

Output: A minimal spanning tree, [V,T].

Assumption: G is connected and has no parallel edges.

```
begin
    U ← {v1}, w ← v1, T ← ∅ ;
    for i = 1 to n, d(i) ← ∞ ;
    create p-1 clones
    (comment: create p-1 additional processes and place them in a wait
              state)
    F ← {(w,v) ε E};
    partition F into mutually exclusive sets F1,...,Fs, s ≤ p;
    for m = 1 to s, fork module forwardscan(m);
    B ← {(u,w) ε E};
```

```
partition B into mutually exclusive sets B_1,...,B_t, t ≤ p;
for m = 1 to t, fork module backwardscan(m);
  while U ≠ V do
    globalmin ← ∞;
    for m = 1 to p, fork module nodescan(m);
    (comment: forks are executed in parallel and processing does not
              continue in the main program until all processes complete
              nodescan)
    T ← T ∪{e(ibest)}, U ← U ∪{w};
    F ← {(w,v) ε E};
    partition F into mutually exclusive sets F_1,..., F_s, s ≤ p;
    for m = 1 to s, fork module forwardscan(m);
    B ← {(u,w) ε E};
    partition B into mutually exclusive sets B_1,..., B_t, t ≤ p;
    for m = 1 to t, fork module backwardscan(m);
  end while
  kill the clones
end
module nodescan(m)
local data: min, x
begin
  min ← ∞
  for i = m to n step p do
    if d(i) < min then min ← d(i), x ← i;
  end for
  lock globalmin
    if min < globalmin then globalmin ← min, ibest ← x, w ← v_x;
  unlock globalmin
end
```

```
module fowardscan(m)
begin
  for all (u,v) ε F_m do;
    if c(u,v) < d(v) then d(v) ← c(u,v), e(v) ← (u,v);
  end for
end
module backwardscan(m)
begin
  for all (u,v) ε B_m do;
    if c(u,v) < d(u) then d(u) ← c(u,v), e(u) ← (u,v);
  end for
end
```

V. COMPUTATIONAL RESULTS WITH PARALLEL ALGORITHMS

Both algorithms of Section IV were coded in FORTRAN for the Balance 21000 located in the Center for Applied Parallel Processing at Southern Methodist University. The Balance 21000 is configured with twenty NS32032 cpu's, 32 Mbytes of shared memory, and 16K user-accessible hardware locks. Each cpu has 8 Kbytes of local RAM and 8 Kbytes of cache. The Balance 21000 runs the DYNIX operating system, a version of UNIX 4.2bsd. DYNIX includes routines to create, synchronize, and terminate parallel processes from C, Pascal, and FORTRAN. More details about the Balance 21000 may be found in [13].

Table 4 gives the computational results with Boruvka's algorithm. The times are wall clock times and are the average for three runs. The first row in each table contains the time for the sequential version of BORUVKA and all other rows contain times for the parallel version. The sequential version is 250 lines of code, while the parallel version required over 400 lines. The speedup for a row is calculated by dividing the best sequential time by the time in that row.

Initially, the parallel code creates the additional processes to be used and requires each of them to build data tables which give the location in virtual memory of all shared data. Once this is done, the processes can be used repeatedly with little system overhead. However, this initial creation and the subsequent killing of those processes at termination can be very expensive for this type of problem. The first column of times includes the creation and process termination time while the second does not. Hence, if a 350 x 350 minimal spanning tree was to be obtained one time, then the best speedup is 2.6 using seven cpu's. If however, this is a subprogram of a larger system, then a 350 x 350 problem can yield a speedup of <u>four</u> on six processors and a speedup of <u>five</u> on ten.

Table 4 About Here

Table 5 gives the computational results with Prim's algorithm. No speedup is achievable for a one-time solution. For environments in which the minimum spanning tree problem is a subproblem, speedups of three and five were obtained on five and ten processors, respectively.

Table 5 About Here

VI. SUMMARY AND CONCLUSIONS

Five computer codes were developed to solve the minimum spanning tree problem on a sequential machine. These codes were computationally compared on both grid graphs and random graphs whose densities varied from 5% to 100%. The implementation of Boruvka's algorithm (see [15, p. 277]) was the best for grid graphs. An implementation of Prim's algorithms using a sparse data representation (see [15, p. 273]) was best for high density random graphs while an implementation of Prim's algorithm using a d-heap (see [19, p. 77]) was best for lower density random problems. Kruskal's algorithm using a quicksort is the most robust of all the implementations, ranking either second or third in all computational tests. Both Boruvka's and Prim's algorithms were parallelized by the method of data partitioning (also called homogeneous multitasking). This involves creating multiple, identical processes and assigning a portion of the data to each processor. For the environment in which a minimal spanning tree problem is a subproblem within a larger system, speedups of <u>five</u> on ten processors were achieved with both Prim's and Boruvka's algorithms. The overhead for parallel processing on the Balance 21000 negates most of the benefits of parallel processing for the first solution of the minimal spanning tree.

REFERENCES

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts (1974).

2. Akl, S., "An Adaptive and Cost-Optimal Parallel Algorithm for Minimum Spanning Trees," Computing, 36 (1986) 271-277.

3. Ali, I., and J. Kennington, "The Asymmetric M-Travelling Salesman Problem: A Duality Based Branch-And-Bound Algorithm," Discrete Applied Mathematics, 13 (1986) 259-276.

4. Christofides, N., Graph Theory: An Algorithmic Approach, Academic Press, New York, New York (1975).

5. Deo, N., and Y. Yoo, "Parallel Algorithms for the Minimum Spanning Tree Problem," Proceedings of the 1981 International Conference on Parallel Processing, IEEE Computing Society Press, (1981) 188-189.

6. Held, M., and R. Karp, "The Travelling Salesman Problem and Minimum Spanning Trees," Operations Research, 18 (1970) 1138-1162.

7. Held, M., and R. Karp, "The Travelling Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming, 1 (1970) 6-25.

8. Knuth, D. E., Sorting and Searching, Addison-Wesley, Reading, Massachusetts (1973).

9. Kwan, S., amd W. Ruzzo, "Adaptive Parallel Algorithms for Finding Minimum Spanning Trees," Proceedings of the 1984 International Conference on Parallel Processing, IEEE Computing Society Press, (1984) 439-443.

10. Lavallee, I., and G. Roucairol, "A Fully Distributed (Minimal) Spanning Tree Algorithm," Information Processing Letters, 23 (1986) 55-62.

11. Lavallee, I., "An Efficient Parallel Algorithm for Computing a Minimum Spanning Tree," Parallel Computing 83, (1984) 259-262.

12. Nath, D., and S. Maheshwari, "Parallel Algorithms for the Connected Components and Minimal Spanning Tree Problems," Information Processing Letters, 14, 1 (1982) 7-11.

13. Osterhaug, A., Guide to Parallel Programming on Sequent Computer Systems, Sequent Computer Systems, Inc., Beaverton, Oregon (1986).

14. Parallel Computers and Computations, Editors J. van Leevwen and J. K. Lenstra, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, (1985).

15. Papadimitriou, C. and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey (1982).

16. Pawagi, S. and I. Ramakrishnan, "An O(log n) Algorithm for Parallel Update of Minimum Spanning Trees," Information Processing Letters, 22 (1986) 223-229.

17. Quinn, M. J., Designing Efficient Algorithms for Parallel Computers, McGraw-Hill, New York, New York (1987).

18. Sedgenwick, R., Algorithms, Addison-Wesley, Reading, Massachusetts (1983).

19. Tarjan, R. E., Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania (1983).

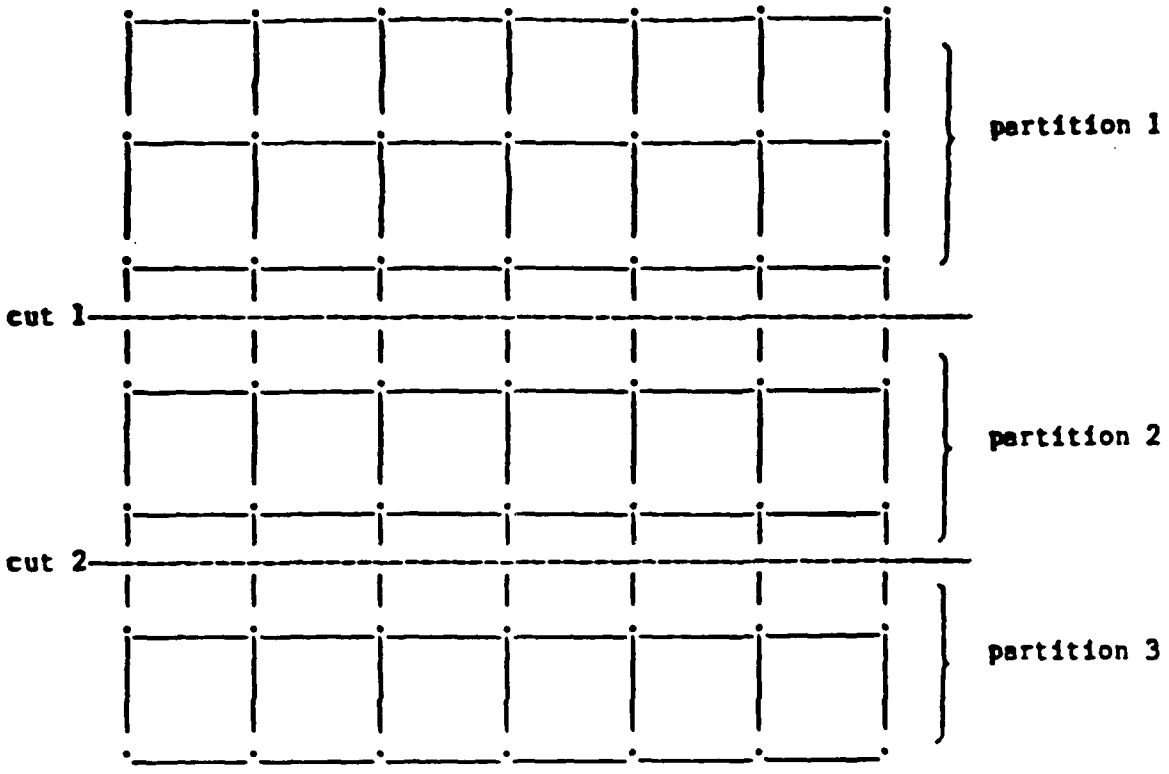


Figure 1. A Three Processor Partitioning of a 7 x 7 Grid Graph.

Table 1. Comparison of Sequential Algorithms on Grid Graphs

(cost range is 0 - 10,000)

Grid Size n x n	Edges	Graph Density	DENSE PRIM	SPARSE PRIM	HEAP PRIM	KRUSKAL	BORUVKA
15 x 15	420	1.7%	1.70	.36	.27	.19	.12
18 x 18	612	1.2%	3.54	.74	.42	.30	.17
20 x 20	760	1.0%	5.43	1.10	.54	.39	.21
24 x 24	1,104	.7%	11.32	2.19	.82	.63	.30
28 x 28	1,512	.5%	21.01	4.09	1.13	.86	.46
30 x 30	1,740	.4%	27.82	5.41	1.37	1.15	.55
Total Time (secs.)			70.82	13.89	4.55	3.52	1.81
Rank			5	4	3	2	1

AD-A185 862 DEVELOPMENT AND EVALUATION OF A CASUALTY EVACUATION MODEL FOR A EUROPEAN.. (U) AIR FORCE OFFICE OF SCIENTIFIC RESEARCH BOLLING AFB DC J L KENNINGTON 3/3

UNCLASSIFIED 10 AUG 87 AFOSR-TR-87-0970 *AFOSR-83-0278 F/G 23/6 NL

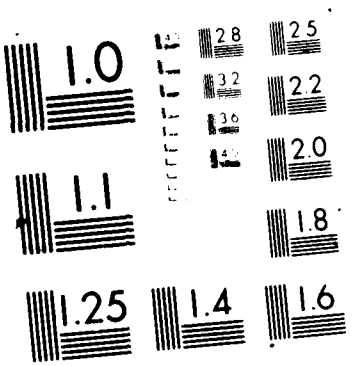

1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
2.5
2.2
2.0
1.8

Table 2. Comparison of Sequential Algorithms on High Density Random Graphs.

(cost range is 0 - 10,000)

Vertices	Edges	Graph Density	DENSE PRIM	SPARSE PRIM	HEAP PRIM	KRUSKAL	BORUVKA
200	19,900	100%	1.39	1.14	1.44	1.52	3.01
200	15,920	80%	1.39	.97	1.22	1.52	1.96
200	11,940	60%	1.39	.79	.99	.96	1.47
200	7,960	40%	1.39	.61	.76	.89	1.02
400	79,800	100%	5.67	4.55	5.42	4.45	12.03
400	63,840	80%	5.69	3.85	4.53	3.58	10.28
400	47,880	60%	5.70	3.13	3.62	2.82	7.26
400	31,920	40%	5.71	2.49	2.68	1.97	4.85
600	179,700	100%	13.28	10.39	11.98	12.38	29.85
600	143,760	80%	13.66	8.79	9.99	14.99	23.72
600	107,820	60%	13.16	7.15	7.99	10.63	17.79
600	71,880	40%	13.02	5.55	5.67	6.05	11.80
Total Time (secs.)			81.45	49.41	56.29	61.76	125.04
Rank			4	1	2	3	5

Table 3. Comparison of Sequential Algorithms on Low Density Random Graphs.

(cost range is 0 - 10,000)

Vertices	Edges	Graph Density	DENSE PRIM	SPARSE PRIM	HEAP PRIM	KRUSKAL	BORUVKA
200	3,980	20%	1.40	.44	.49	.50	.52
200	1,990	10%	1.40	.36	.39	.40	.35
200	995	5%	1.39	.32	.32	.35	.17
400	15,960	20%	5.66	1.75	1.62	1.47	2.46
400	7,980	10%	5.71	1.40	1.12	1.53	1.30
400	3,990	5%	5.72	1.21	.86	1.20	.72
600	35,940	20%	13.04	3.94	3.39	3.99	6.02
600	17,970	10%	13.04	3.05	2.14	2.89	2.86
600	8,985	5%	13.07	2.73	1.50	2.12	1.52
Total Time (secs.)			60.43	15.20	11.83	14.45	15.92
Rank			5	3	1	2	4

Table 4. Parallel Boruvka on 350 x 350 Grid Graph
|V| = 122,500 |E| = 244,300
(cost range is 0 - 100,000)

cpu's	PARALLEL BORUVKA (includes process creation)		PARALLEL BORUVKA (excludes process creation)	
	time	speedup	time	speedup
1+	98.21	1.00	98.21	1.00
1*	112.57	.87	103.86	.95
2	66.93	1.47	57.49	1.71
3	50.26	1.95	40.92	2.40
4	40.25	2.44	29.95	3.28
5	39.00	2.52	26.52	3.70
6	38.69	2.54	23.45	4.19
7	37.70	2.60	21.62	4.54
8	40.98	2.40	21.58	4.55
9	42.49	2.31	20.85	4.71
10	41.30	2.38	17.52	5.61

\+ best sequential BORUVKA code
* parallel code run with a single processor

Table 5. Parallel Prim on G = [V,E] with |V| = 900 and |E| = 404,550

(cost range is 0 - 100,000)

cpu's	PARALLEL PRIM (includes process creation)		PARALLEL PRIM (excludes process creation)	
	time	speedup	time	speedup
1+	24.88	1.00	24.88	1.00
1*	27.09	.92	26.98	.92
2	23.35	1.07	15.12	1.65
3	22.63	1.10	10.84	2.30
4	25.31	.98	8.74	2.85
5	28.43	.88	7.39	3.37
6	31.54	.79	6.62	3.76
7	36.51	.68	6.03	4.13
8	41.08	.61	5.62	4.43
9	46.04	.54	5.30	4.69
10	50.54	.49	5.02	4.96

\+ best sequential PARALLEL PRIM code

* parallel code run with a single processor

END

12-87

DTIC